НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

«На правах рукопису»

УДК _____ ___

«До захисту допущено»

Завідувач кафедри

_____ *С.Г. Стіренко*

(підпис) (ініціали, прізвище)

“ ” _____ 20***20*** р.

Магістерська дисертація

зі спеціальності: *121. Інженерія програмного забезпечення*

(код та назва напряму підготовки або спеціальності)

Спеціалізація: *121. Інженерія програмного забезпечення комп'ютерних систем*

на тему: Багатомодульна платформа трейдингу транспортними засобами

Виконав: студент 6 курсу, групи ІП-94мп

(шифр групи)

Альохін Кирило Валерійович

(прізвище, ім'я, по батькові) (підпис)

Науковий керівник проф., д.т.н., проф., Жабін В. І.

(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант _____

(назва розділу) (посада, вчене звання, науковий ступінь, прізвище, ініціали) (підпис)

Рецензент _____

(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.

Студент _____

(підпис)

Київ – 2020 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет (інститут) *Інформатики та обчислювальної техніки*
(повна назва)

Кафедра *Обчислювальної техніки*
(повна назва)

Рівень вищої освіти – другий (магістерський) за освітньо-професійною програмою
Спеціальність *121. Інженерія програмного забезпечення*
(код і назва)

Спеціалізація *121. Інженерія програмного забезпечення комп'ютерних систем*
(код і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри
С.Г. Стіренко
(підпис) (ініціали, прізвище)
« » _____ 20***20*** р.

ЗАВДАННЯ
на магістерську дисертацію студенту
Альохіну Кирилу Валерійовичу
(прізвище, ім'я, по батькові)

1. Тема дисертації *Багатомодульна платформа трейдингу транспортними засобами*
Науковий керівник дисертації Жабін Валерій Іванович, *д.т.н., проф.*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «*26*» *жовтня 2020* р. № *3133-с*
2. Строк подання студентом дисертації *24 листопада 2020*
3. Об'єкт дослідження *процес налаштування платформи під конкретного клієнта для продажу транспортних засобів.*

4. Предмет дослідження *метод динамічного розгортання платформи під кожного клієнта.*

5. Перелік завдань, які потрібно розробити: *Опис предметної області, дослідження існуючих систем, архітектура та технічний дизайн системи, реалізація додатку у вигляді незалежних мікросервісів.*

Консультанти розділів дисертації:

Розділ	Консульт	Підпис, дата	
		Завдання видав	Завдання прийняв
Розділ 1. Огляд існуючих рішень			
Розділ 2. Вибір засобів розробки і архітектури проекту			
Розділ 3. Реалізація додатку			
Розділ 4. Інструкція користувача і тестування додатку			
Розділ 5. Розробка стартап-проекту			

7. Дата видачі завдання _____

Календарний план

№ п/п	Найменування етапів дипломного проекту (роботи)	Строк виконання етапів роекту(роботи)	Примітки
1..	*Затвердження теми роботи*	*18.12.2019*	
2.	*Вивчення та аналіз завдання*	*20.05.2020-02.06.2020*	
3.	*Розробка загальної архітектури та вибір технологій проекту*	*03.06.2020-20.07.2020*	
4.	*Розробка структур окремих підсистем*	*21.07.2020-29.07.2020*	
5.	*Програмна реалізація системи*	*30.07.2020-30.08.2020*	
6.	*Оформлення пояснювальної записки*	*01.09.2020-02.11.2020*	
8.	*Передзахист*	*24.11.2020*	
9.	*Захист*	*16.12.2020*	

Студент _____ _____
(підпис) (ініціали, прізвище)

Науковий керівник дисертації _____ _____

1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

Дане технічне завдання поширюється на розробку інтернет платформи для трейдингу транспортними засобами.

Область використання: процеси пошуку та продажу майна, налаштування рішення для бізнесу.

2. ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на виконання роботи магістерської дисертації, утвердженого кафедрою обчислювальної техніки Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

3. ЦІЛЬ ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для надання незалежної інтернет платформи для трейдингу транспортними засобами із можливістю динамічної конфігурації та розгортання додатку під конкретне бізнес рішення.

4. ДЖЕРЕЛО РОЗРОБКИ

Джерелом розробки є технічне завдання згідно якого потрібно реалізувати систему для продажу транспортних засобів із можливістю динамічної конфігурації платформи.

5. ТЕХНІЧНІ ВИМОГИ ДО ПРОГРАМИ

5.1. Вимоги до розроблюваного продукту

Додаток повинен забезпечити такі основні функції:

- Повинен надавати можливість налаштувати основні компоненти системи.
- Налаштування динамічних структур даних та їх наступна утилізація.
- Повинен надавати можливість користувачу зареєструватись та приєднатись у роль на основі конфігурації.
- Повинен надавати можливість завантажувати своє оголошення на дошку згідно структур які були задані інженером платформи.
- Користувач повинен мати можливість керувати оголошеннями, здійснювати переходи по статусам згідно конфігурації.

Розробку системи виконати за допомогою мови програмування Java, із використанням веб фреймворку Spring. Оскільки була обрана мікросервісна архітектура, для реалізації цієї цілі можемо використати технологію Spring Cloud, яка забезпечить усіма необхідними технологічними рішеннями для забезпечення Cloud інфраструктури.

5.2. Вимоги до програмного забезпечення

Програмне забезпечення повинно відповідати таким вимогам:

- швидкість роботи та конфігурації даних користувача;
- простота експлуатації;
- надійність.

6. ВИМОГИ ДО ДОКУМЕНТАЦІЇ

- Титульний лист.
- Лист завдання.
- Анотації.
- Опис альбому.
- Технічне завдання.
- Зміст пояснювальної записки.
- Перелік умовних позначень.
- Пояснювальна записка.
- Висновки.
- Список літератури.
- Лістинг програми.

7. ЕТАПИ РОЗРОБКИ

Етап розробки	Дата
Вивчення літератури	
Складання і узгодження технічного завдання	
Аналіз структури	
Програмна реалізація продукту	
Тестування програми	
Налагодження і виправлення помилок	
Оформлення документації дипломної роботи	

РЕФЕРАТ

на магістерську дисертацію

виконану на тему: Багатомодульна платформа трейдингу транспортними засобами

студентом: Альохіним Кирилом Валерійовичем

Робота складається із вступу та п’яти розділів. Загальний обсяг роботи: 90 аркуші основного тексту, 48 ілюстрації, 17 таблиць. При підготовці використовувалася література з 8 різних джерел.

Актуальність. У наш час, досить актуальною темою є надання специфічних рішень, для торгівлі товарами, під потреби кожного клієнта. Моє рішення надає, представникам малого та середнього бізнесу, можливість динамічної конфігурації та розгортання платформи для трейдингу транспортних засобів. За рахунок цього, користувачі можуть купувати та продавати власну техніку.

Мета і завдання дослідження. Метою магістерської дисертації є надання бізнесу автоматизованої інтернет платформи для менеджменту, конфігурації та розгортання специфічного під кожного клієнту рішення для трейдингу. За рахунок цього, бізнес зможе автоматизувати усі свої процеси по продажу техніки, із забезпеченням усіх необхідних побажань та специфік ведення продажів.

Об’єкт дослідження – процес налаштування платформи під конкретного клієнта для продажу транспортних засобів.

Предмет дослідження – метод динамічного розгортання платформи під кожного клієнта.

Методи досліджень. Для досягнення поставлених в магістерській роботі задач використано методи побудови програмного додатку на основі мікросервісної архітектури.

Наукова новизна одержаних результатів роботи полягає у запропонованому методі, що дає можливість розгортання інтернет платформи для продажу транспортних засобів, із можливістю динамічного налаштування її компонентів під потреби кожного клієнта.

Особистий внесок здобувача. Магістерське дослідження є самостійно виконаною роботою, в якій відображено особистий авторський підхід та особисто отримані теоретичні та прикладні результати, що відносяться до проектування та дизайну архітектури програмного коду додатку на основі мікросервісної архітектури, які працюють у хмарному середовищі. Формулювання мети та завдань дослідження проводилось спільно з науковим керівником.

Практична цінність. На основі проведених досліджень про актуальність додатків для торгівлі транспортом, був розроблений програмний продукт платформу, яка надає рішення для трейдингу транспортними засобами із забезпеченням функціоналу специфічним під потреби кожного клієнта.

Конференції:

Ключові слова: Мікросервісна архітектура, платформа, трейдинг транспортними засобами, хмарні рішення, динамічна конфігурація.

ABSTRACT

on master’s thesis

made on the topic: Multimodular vehicle trading platform

student: Alokhin Kyrylo Valeriovych

The work consists of an introduction and five sections. Total work: 90 sheets of the main text, 48 illustrations, 17 tables. Literature from 8 different sources was used in the preparation.

Topicality. Nowadays, a very important topic is the provision of specific solutions for trade in goods for the needs of each client. My solution gives small and medium-sized businesses the opportunity to dynamically configure and deploy a vehicle trading platform. Due to this, users can buy and sell their own equipment.

The purpose and objectives of the study. The purpose of the master's thesis is to provide the business with an automated Internet platform for management, configuration and deployment of a specific trading solution for each client. Due to this, the business will be able to automate all its processes for the sale of equipment, providing all the necessary wishes and specifics of sales.

The object of research - the process of setting up a platform for a specific customer to sell vehicles.

The subject of research - the method of dynamic deployment of the platform for each client.

Research methods. To achieve the objectives set in the master's thesis, methods of building a software application based on microservice architecture were used.

The scientific novelty of the obtained results about proposed method that allows the deployment of an Internet platform for the sale of vehicles, with the ability to dynamically adjust its components to the needs of each customer.

The study allows for dynamic configuration of the platform and the provision of different solutions, depending on business requirements.

Personal contribution of the applicant. The master's research is a self-performed work, which reflects the personal author's approach and personally obtained theoretical and applied results related to the design and engineering architecture of the application code based on microservice architecture, working in a cloud environment. The formulation of the purpose and objectives of the study was carried out jointly with the supervisor.

Practical value. Based on research on the relevance of applications for transport trade, a software product platform has been developed that provides solutions for trading vehicles with the provision of functionality specific to the needs of each client.

Conferences:

Keywords: Microservice architecture, platform, vehicle trading, cloud solutions, dynamic configuration.

ПОЯСНЮВАЛЬНА ЗАПИСКА

до магістерської дисертації

на тему: Багатомодульна платформа трейдингу транспортними засобами

ЗМІСТ

СПИСОК СКОРОЧЕНЬ

API (Application program interface) Програмний інтерфейс.

B2C (Business to client) Бізнес для клієнта

B2B (Business to business) Бізнес для бізнесу.

DDD (Domain Driven Design) Предметно орієнтоване програмування.

ERD (Entity Relationship Diagram) Діаграма моделей залежностей.

JDK (Java Development Kit) Засоби розробки Java.

KISS (Keep it simple, stupid) Архітектурний принцип про спрощення системи.

OS (Operating system) Операційна система.

RTD (Real time data) Дані в системі реального часу.

UI (User interface) Інтерфейс користувача.

UX (User experience) Досвід користувача.

ВСТУП

У зв'язку із стрімким збільшенням технологій, малий та середній бізнес зацікавлені у інтеграції своїх систем із інтернет комунікаціями. В сучасному світі, без цього абсолютно ніяк не обійтись. Навіть елементарне замовлення таксі, зараз, настільки автоматизований процес, що може бути здійснений за рахунок двох кліків на свому смартфоні. Механізм продажу автомобільної техніки не повинен бути виключенням, і кожен клієнт повинен мати змогу інтегрувати свій бізнес із інтернет рішенням. Моя платформа надає йому цю можливість.

Основна ідея додатку це надання бізнесу автоматизованої інтернет платформи для менеджменту, конфігурації та розгортання специфічного під кожного клієнту рішення для трейдингу. За рахунок цього, бізнес зможе автоматизувати усі свої процеси по продажу техніки, із забезпеченням усіх необхідних бажань та специфік ведення продажів.

Отже, з точки зору продукту - це буде єдиний продукт з можливістю кастомізації в режимі реального часу.

РОЗДІЛ 1
ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

1.1. Загальні відомості про платформу трейдингу автомобільної техніки

Автомобільна промисловість знаходиться на початку найбільшого перетворення. Нові бізнес-моделі, що змінюються галузеві правила і вибух цифрових технологій і хмарних рішень, а також зростання числа електромобілів руйнують автомобільний ландшафт. Моє рішення надає користувачу а також клієнту незалежну інтернет платформу для конфігурації та менеджменту усіх необхідних компонентів додатку для продажу автомобілів.

Більшість клієнтів котрі хочуть зробити бізнес продажу автівок користуються готовими веб сайтами для розміщення предметів своєї торгівлі. Проте, при такому підході можна виділити деякий набір недоліків:

- Обмежений функціонал - в більшості веб-додатків функціонал для продажу є обмеженим і розробники рідко надають оновлення в інтересах потенційних клієнтів-користувачів
- Комісія з кожної купівлі
- Пряма залежність від існуючих рішень

Тому клієнт хотів би бачити певний набір функціналу для розгортання власного інтернет порталу з можливістю динамічної конфігурації контенту, основних компонентів, платіжних систем та інших незалежних функціональних модулей.

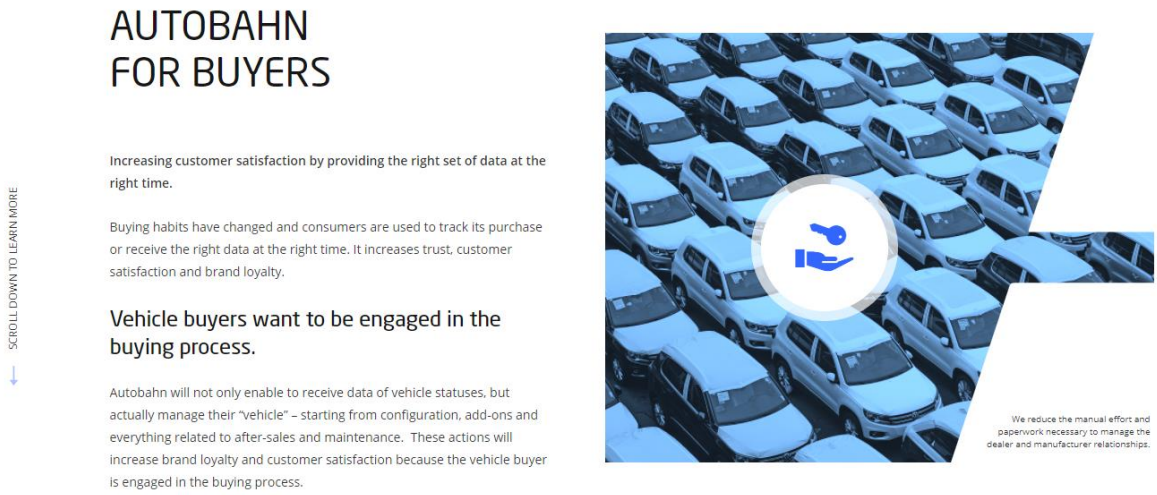

Рис. 1.1. Приклад звичайного веб сайту для продажу автомобілів [9]

Особливою популярністю користується у власників малого та середнього бізнесу, індивідуальних підприємців, представників різного роду спеціальностей, а також особистостей з творчого середовища. Для роботи з моєю платформою не потрібно якихось спеціальний знань або навичок програміста. Збірка платформи здійснюється в візуальному редакторі, його інтерфейс може освоїти і школяр, і людина у віці. Вбудовані шаблони замінюють ручне втручання та додають швидкості при розгортанні платформи. Кожен з них наповнений відповідним тематиці демо-контентом, що виявиться для більшості людей відмінною підказкою при розробці.

1.2. Порівняння існуючих рішень

1.2.1. Autobahn-tech

Дане рішення допомагає автомобільним компаніям в цифровому перетворенні. Вони надають інтернет платформу для реструктуризації бізнес-операцій і зв'язку між виробниками, імпортерами та дилерами. Основною перевагою даного рішення є збереження усіх незалежних компонентів системи та даних в хмарному середовищі.

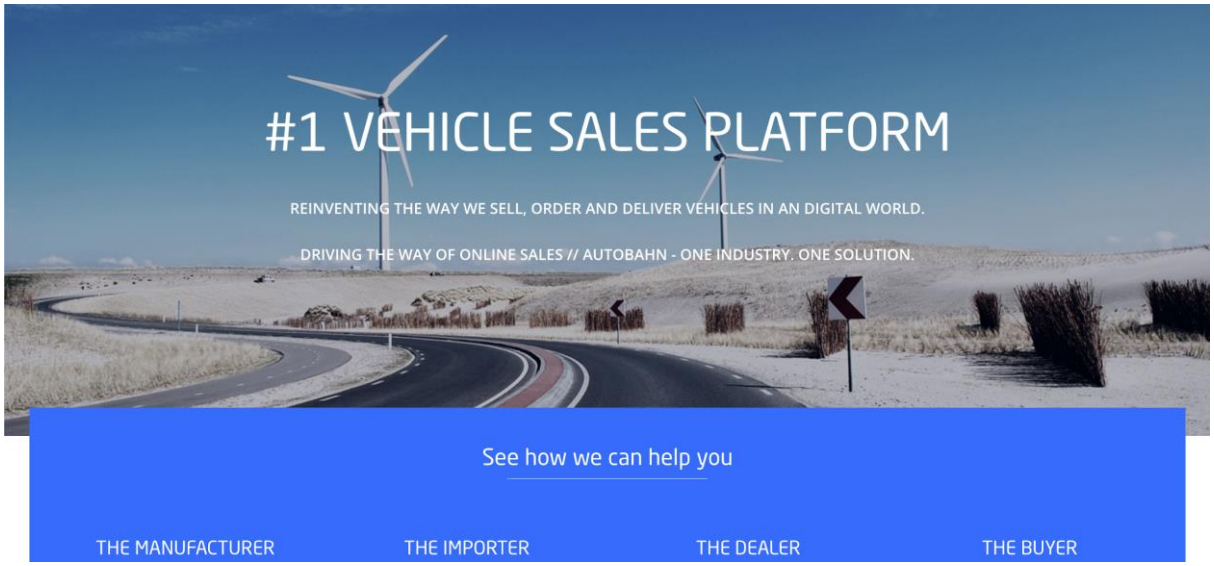

Рис. 1.2. Візуальний вигляд сайту додатку Autobahn-tech [9]

1.2.2. Wizzle

Wizzle був створений, щоб зробити весь процес продажу простіше. Наша безкоштовна онлайн-платформа дозволяє безкоштовно рекламувати ваш автомобіль. Зібравши всю інформацію про поточний стан вашого автомобіля, покупець може зробити вам реалістичну пропозицію.

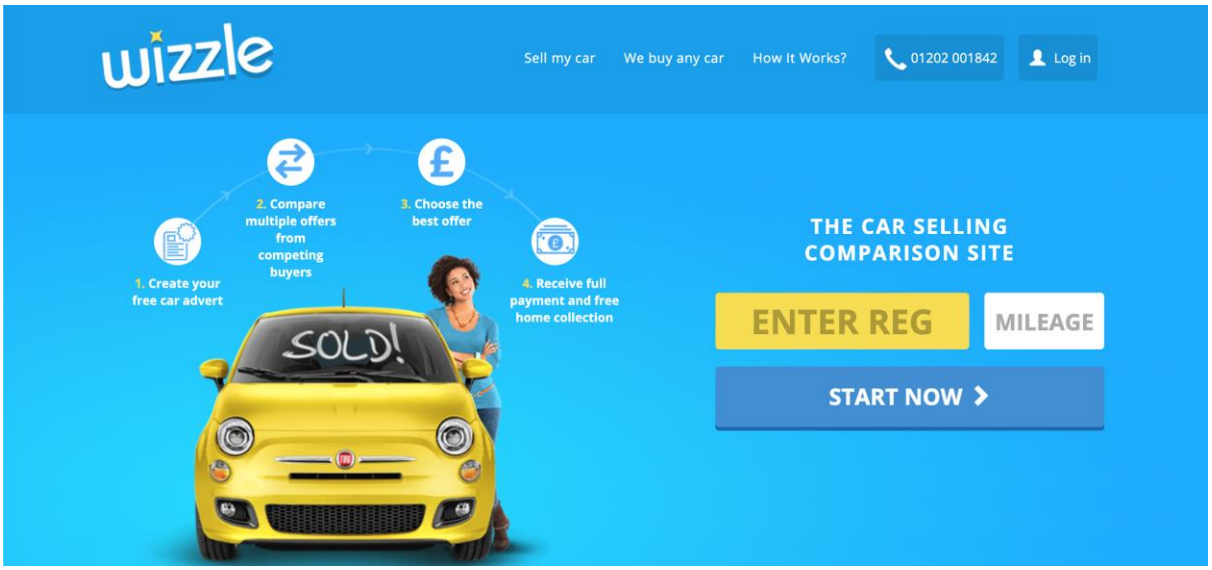

Рис. 1.3. Візуальний вигляд додатку Wizzle [10]

Wizzle використовують те ж саме передове програмне забезпечення, яке дилери використовують для професійної оцінки транспортних засобів. Просто ввівши свої дані в додаток Wizzle уся інформація про вашому автомобілі передається в нашу мережу тисяч автомобільних трейдерів. Будь-які покупці, які зацікавлені в проведенні торгів, запропонують свої кращі ціни.

1.2.3. Carvago

Універсальна платформа дозволяє створювати однаково потужні інтернет-магазини для продажу автівок. Система збалансована і дуже функціональна, надає повний доступ до html / css / php коду, FTP і SEO налаштувань. У плані багатозадачності у Carvago конкурентів не існує. Орієнтується на більш-менш досвідчених користувачів, новачкам спочатку буде трохи складно освоїтися.

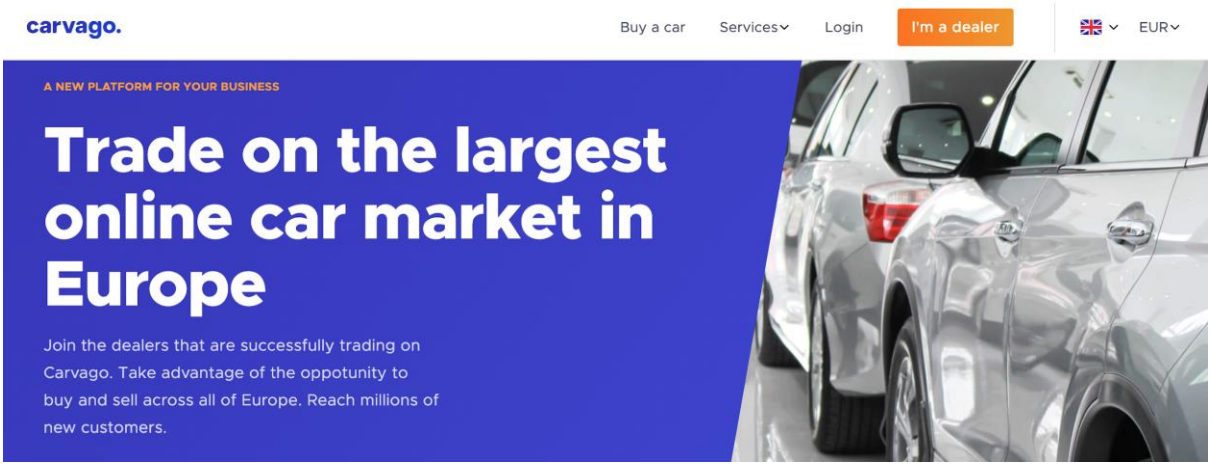

Рис. 1.4. Візуальний вигляд додатку Carvago [11]

1.2.4. ADESA

ADESA продають високоякісні старі автомобілі виключно і безпосередньо автодилерам і трейдерам. Всі автомобілі поставляються з якісними описами автомобілів і перевірками.

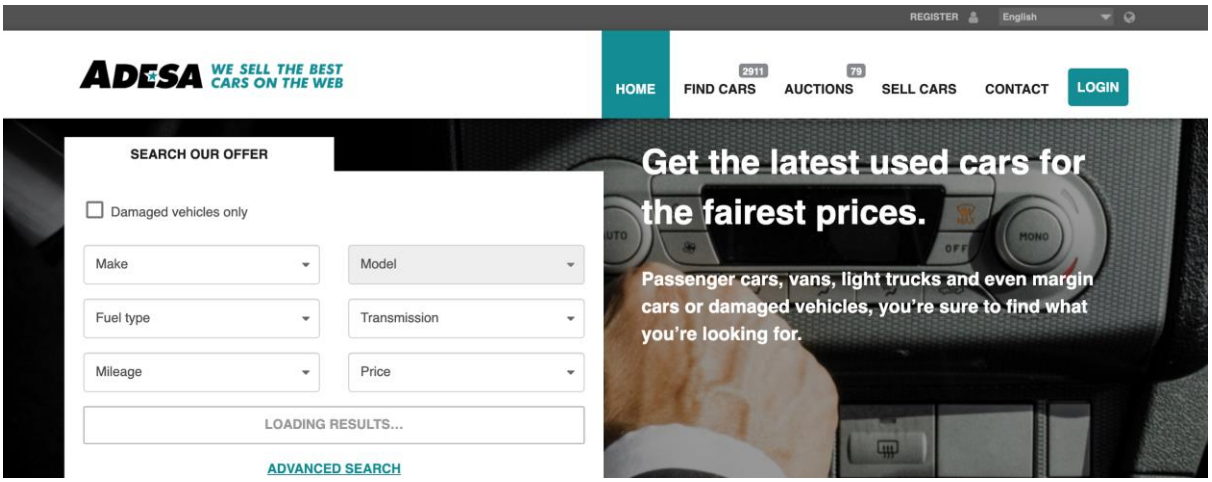

Рис. 1.5. Візуальний вигляд додатку ADESA [12]

Веб сайт має дуже логічну панель управління, що значно полегшує налаштування та конфігурацію платформи під клієнта. Під кожен тип сайтів і завдань тут є окремий додаток. Який складається з основного модуля, загальних налаштувань а також декількох плагінів. Також завантажена велика кількість шаблонів які замінюють ручну авто конфігурацію.

1.2.5. ZigWheels

Основним функціоналом ZigWheels розповсюдження інформації по продажу автівок різним дилерам. Таким чином сервіс робить ще один крок, щоб інформувати покупців про справжню ринкову вартість автомобіля. Користувач зможе обрати вдалий час для покупок якщо йому потрібно

розширити область пошуку.

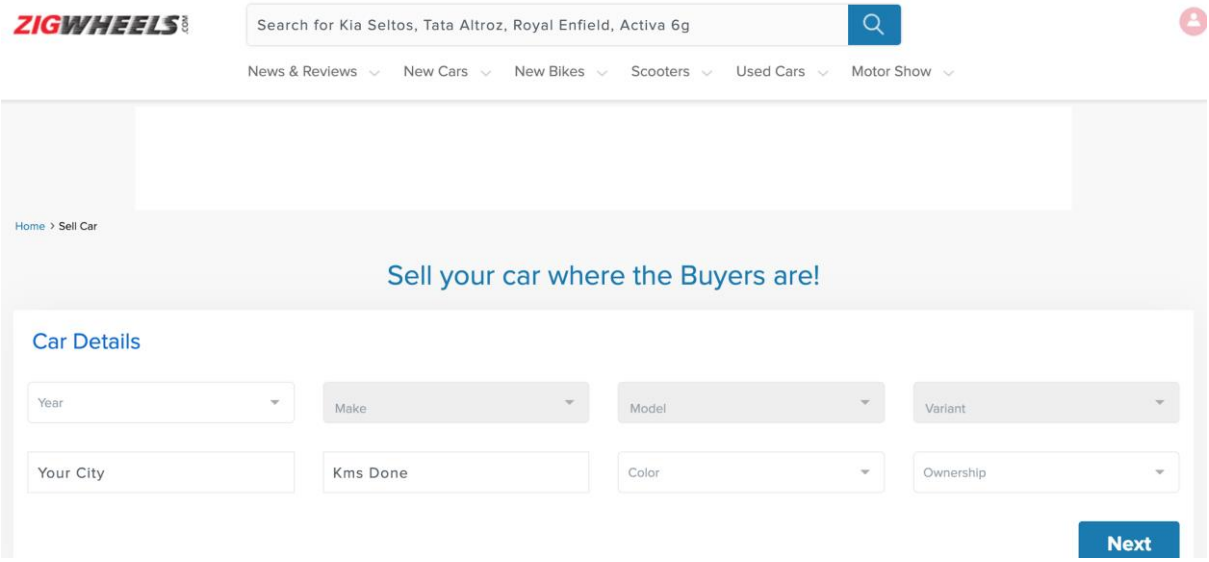

Рис. 1.6. Візуальний вигляд додатку ZigWheels [13]

Правити конфігурацію можна як завгодно, адже ZigWheels - це повноцінна CMS, встановлена на хостинг і заздалегідь оптимізована. Надається повний доступ до файлів сайту по FTP. Крім того, правити код модулів і сторінок можна прямо з панелі управління. Є й штатні налаштування кастомізації - колірна гамма, фони, кнопки, структура.

1.2.6. SWAP MOTORS

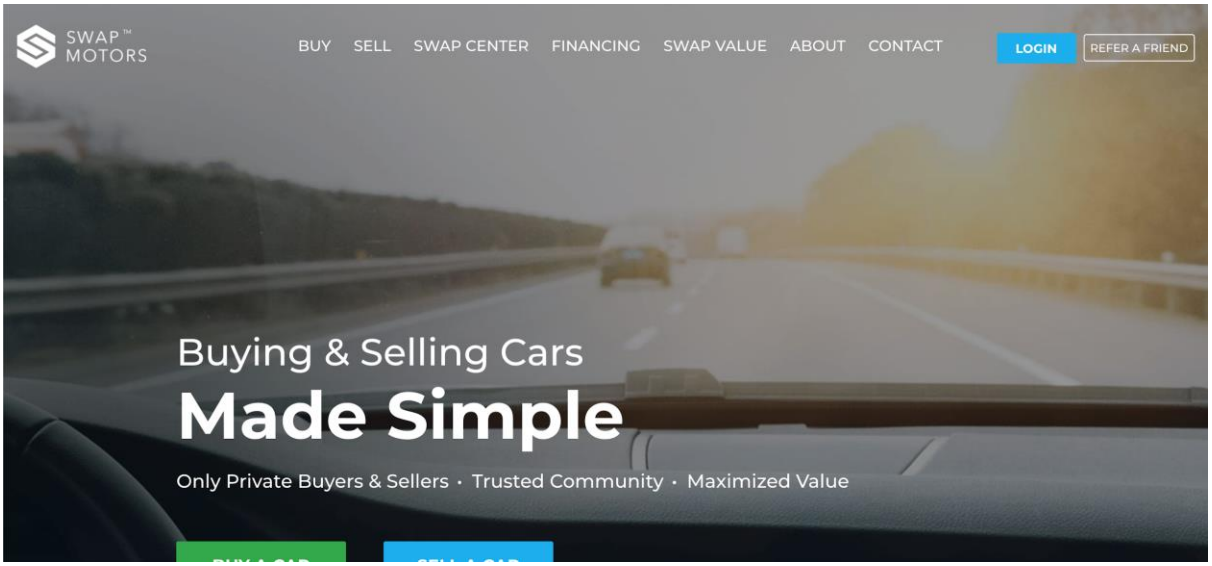

Рис. 1.7. Візуальний вигляд додатку SWAP MOTORS [14]

SWAP MOTORS - надають весь необхідний перелік функціоналу для налаштування власного бізнес рішення. Внутрішній двигун добре пристосований для створення дуже великих магазинів. Тут є навіть можливість синхронізації з «1С: Управління торгівлею», не кажучи вже за імпорт / експорт товарів через табличні формати файлів. Хостинг і двигун досить швидко працюють, і навіть при великому напливі відвідувачів сайти SWAP MOTORS досить добре справляється.

1.2.7. CarWow

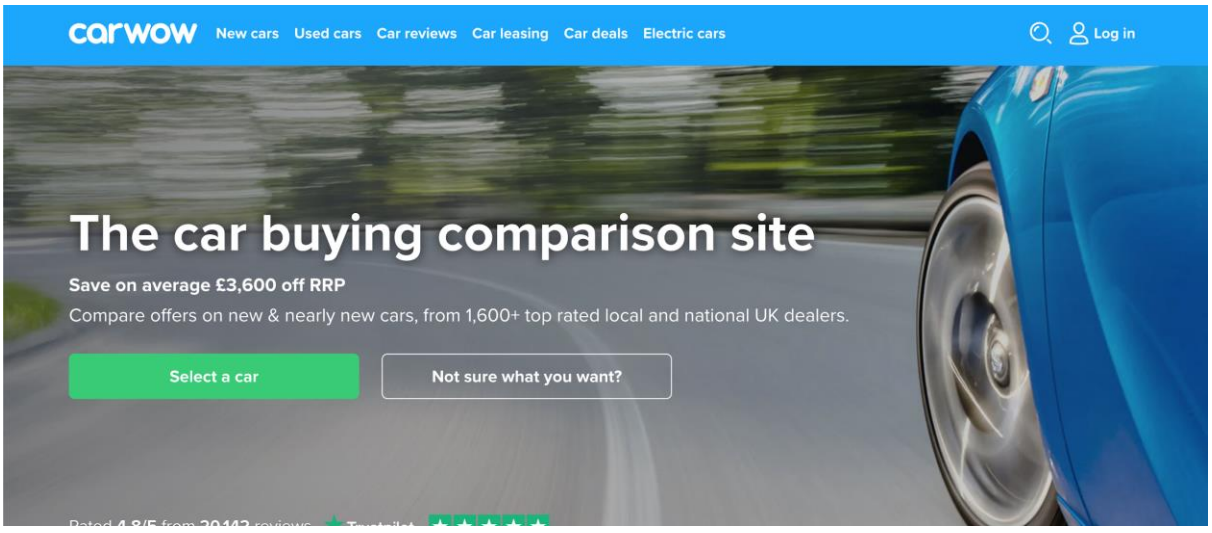

Рис. 1.8. Візуальний вигляд додатку CarWow [15]

CarWow - чудове рішення для тих хто хочу налаштувати рішення швидко та надати його користувачу. Також тут присутня система налаштування аукціонів та динамічне розгортання їх на різних площадках.

1.2.8. CarSome

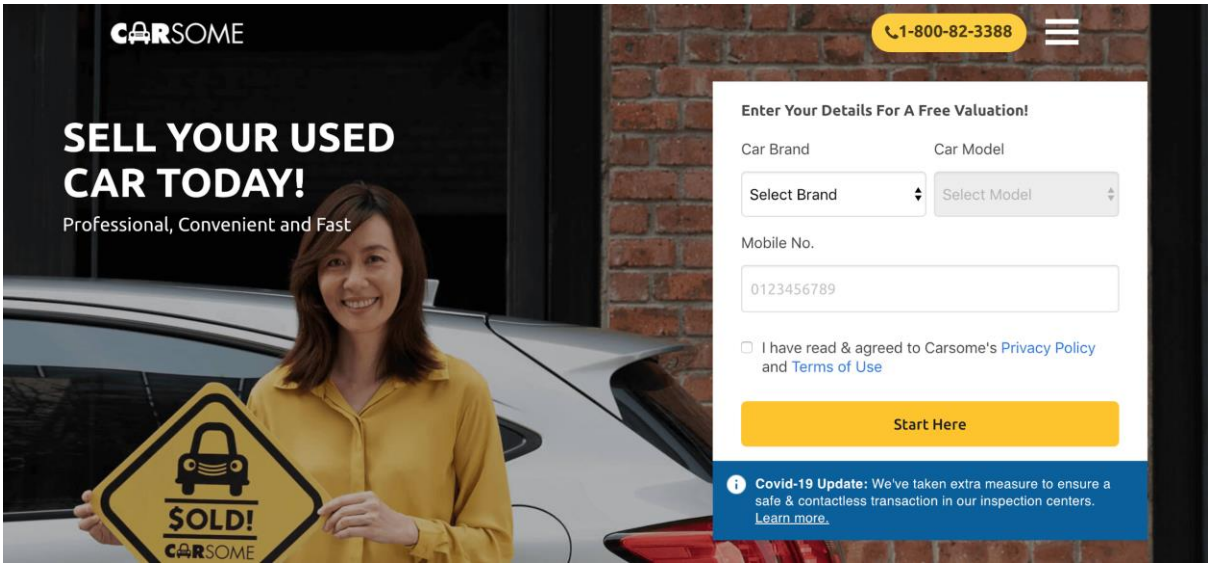

Рис. 1.9. Візуальний вигляд додатку CarSome [16]

CarSome пропонує можливість досліджувати ціни нових автомобілів і балансову вартість уживаних автомобілів. Ви можете сортувати по маркам, миль за галон і діапазонами цін (від 15 до 75 тис. Доларів). Вони також пропонують «найкращі ціни» на основі розташування, зручний інструмент порівняння автомобілів, калькулятор, систему платежів і посилання на Geico, яку можна використовувати для отримання страхового пропозиції.

Сертифікований центр вживаних автомобілів дозволяє вам шукати ціни «CPO» для автомобіля, який ви хочете, перераховує варіанти гарантії і надає роздруківки для цього конкретного автомобіля у вашій поштовій індексі.

Також розробники надають чудовий інструмент порівняння чотирьох автомобілів. Більшість інструментів дозволяють побачити паралельні основи, такі як ціна і MPG. Цей інструмент, проте, також покаже вам зображення інтер'єру і екстер'єру автомобіля, а також п'ятирічний прогноз витрат. До них відносяться паливо, страховка, податки і збори і технічне обслуговування.

Висновок до першого розділу

Провівши аналіз багатьох найпопулярніших інтернет платформ для розгортання своєї бізнес моделі купівлі/продажу автівок можемо виділити майбутній функціонал мого додатку:

- Реєстрація користувача
- Логін користувача
- Налаштування основних компонентів платформи
- Динамічна конфігурація функціоналу
- Розгортання системи
- Взаємодія системи з користувачем

РОЗДІЛ 2
ТЕХНОЛОГІЧНІ РІШЕННЯ ТА АРХІТЕКТУРА СИСТЕМИ

2.1. Вибір мови, технологічної платформи

Для реалізації додатку я обрав мову програмування Java. Щоб реалізувати серверну частину я використав Spring Framework [1] [2] та його підпроекти Spring Cloud [5] для забезпечення мікросервісної архітектури. Кожен із мікросервісів підключений до загальної екосистеми мікросервісів, реєструючись через загальний Discovery сервіс.

2.1.1. Spring Cloud

Розробка, розгортання і експлуатація хмарних додатків повинні бути такими ж простими, як (якщо не простіше, ніж) локальних додатків. Це і повинно бути керівним принципом будь хмарної платформи, бібліотеки або інструменту. Spring Cloud [5] [25] - бібліотека з відкритим вихідним кодом - спрощує розробку додатків JVM для хмари. З його допомогою додатка можуть підключатися до сервісів і легко знаходити інформацію про хмарної середовищі в декількох хмарах, таких як Cloud Foundry і Heroku. Крім того, ви можете поширити його на інші хмарні платформи і нові сервіси.

Програми, що працюють з архітектурою мікросервісів, спрямовані на спрощення розробки, розгортання і обслуговування. Розкладена природа програми дозволяє розробникам зосередитися на одній проблемі за раз. Покращення можуть бути внесені без впливу на інші частини системи.
З іншого боку, інші проблеми виникають, коли ми застосовуємо мікросервісний підхід:

- Зовнішня конфігурація, так що вона гнучка і не вимагає перебудови служби при зміні
- Виявлення послуг

- Приховування складності служб, розгорнутих на різних хостах

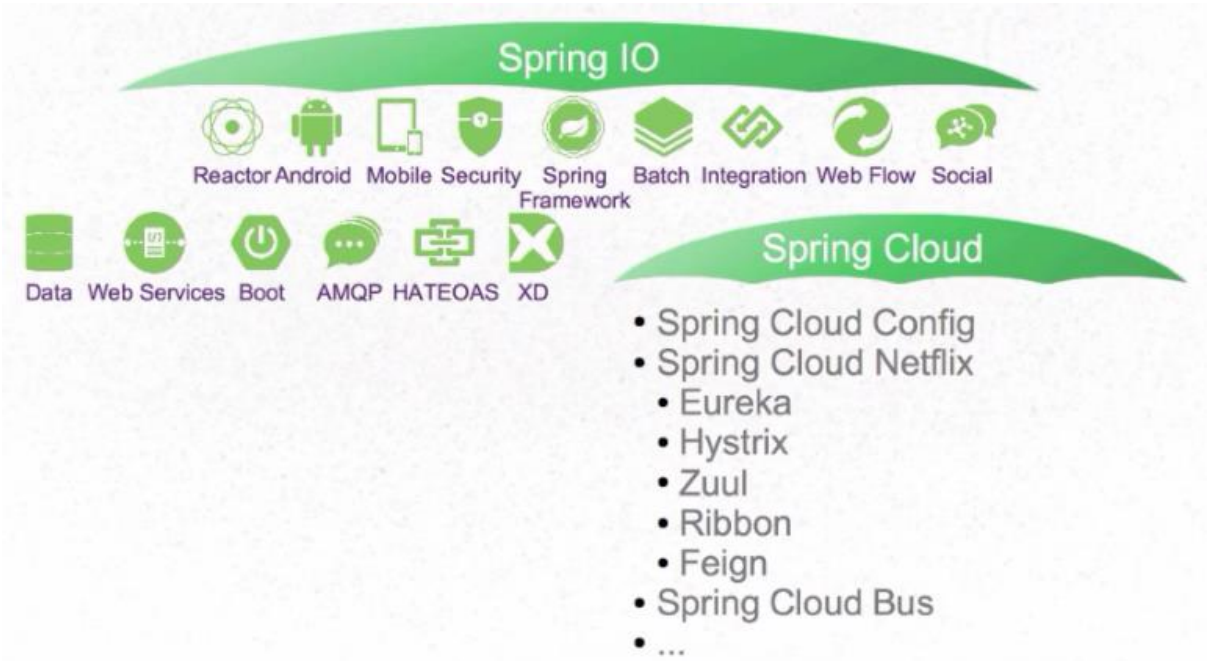

Рис. 2.1. Опис компонентів з яких складається Spring Cloud

Одним з багатьох переваг запуску додатка в хмарі є легка доступність різних сервісів. Замість управління обладнанням, установкою, роботою, резервним копіюванням і т. д. Ви просто створюєте і прив'язуєте сервіси одним натисканням кнопки або командою оболонки.

Як додатки отримують доступ до цих послуг? Наприклад, якщо до вашого додатком прив'язана реляційна база даних, вам потрібно буде створити об'єкт DataSource на основі цієї служби. Ось де Spring Cloud допомагає. Він видаляє всю роботу, необхідну для доступу і налаштування з'єднувачів служб, і дозволяє зосередитися на використанні цих служб. Він також надає інформацію про екземпляр додатка (адреса хоста, порт, ім'я та т. Д.).

Spring Cloud робить все це незалежно від хмари, використовуючи концепцію Cloud Connector. Хоча він надає реалізації для Cloud Foundry і Heroku, ви (або постачальник хмарних обчислень) можете розширити його до інших хмар, запровадивши інтерфейс і скориставшись перевагами іншої частини

бібліотеки. Потім просто додайте бібліотеку, яка містить розширення, в шлях до класів вашого застосування; немає необхідності розбирати і збирати Spring Cloud.

Spring Cloud також визнає, що не може обслуговувати кожен сервіс в кожному хмарі. Таким чином, незважаючи на те, що він підтримує багато поширених послуги з коробки, він дозволяє вам (або постачальнику послуг) розширювати свої функціональні можливості для інших служб. Так само, як розширення для інших хмар, ви додаєте jar, що містить розширення вашого сервісу, в шлях до класу вашого застосування.

Нарешті, він має спеціальну підтримку додатків Spring (в окремому модулі) із додатками Spring Boot, в формі підтримки конфігурації Java і XML, а також надає властивості додатків і служб в зручній для використання формі. Це єдиний модуль в Spring Cloud, який залежить від Spring. Інші постачальники платформ можуть надавати конкретну підтримку для своїх платформ аналогічним чином.

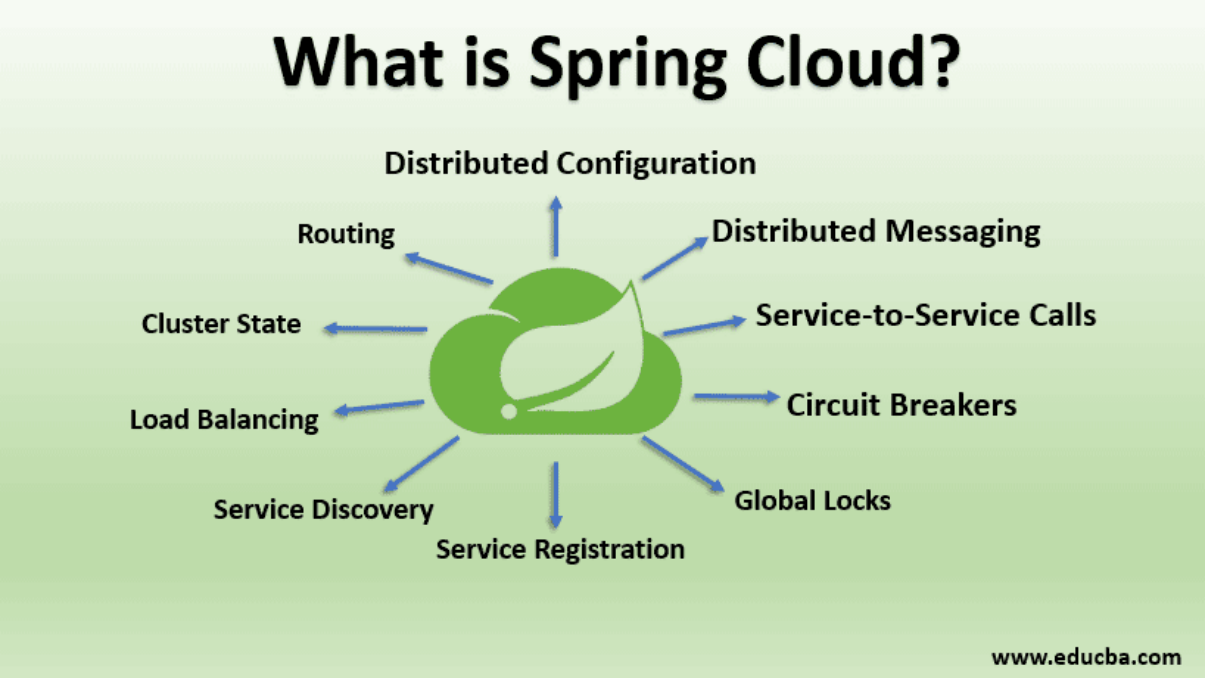

Рис. 2.2. Які задачі вирішує Spring Cloud

Spring Cloud орієнтований на надання необхідних функціональних можливостей. Існують три шляхи розширення:

- Конфігурація Java і XML для Spring-додатків:
- Розширення хмарної платформи: Використовуючи Cloud Connector, можливе розширення Spring Cloud для інших хмарних платформ
- Інформація про сервіс та розширення Connector: Spring Cloud дозволяє підключатися до різних типів сервісів так довго, поки є можливість отримувати інформацію про підключення з додатків операційної системи і при необхідності, конвертувати в сервісний роз'єм

Проект складається з:

- Spring Cloud Core: Основна бібліотека, яка не залежить від хмар і Spring
- Spring Cloud Service Connector для Spring: Бібліотека, що надає коннектори для створення javax.sql.DataSource об'єкти і різні "фабрики" для Spring-проектів
- Cloudfoundry Connector: Конектор для Cloud Foundry
- З'єднувач Heroku: Конектор для Heroku
- Наданий користувачем коннектор служби Cloudfoundry: Розширення Cloud Foundry Connector для підтримки доступних сервісів

2.1.2. Service Discovery (Eureka)

Виявлення служби на стороні клієнта дозволяє службам знаходити і обмінюватися даними один з одним без жорсткої вказівки імені вузла та порту. Єдина «фіксована точка» в такій архітектурі складається з реєстру служб, в якому кожна служба повинна реєструватися.

Недоліком є те, що всі клієнти повинні реалізувати певну логіку для взаємодії з цією фіксованою точкою. Це передбачає додатковий обмін даними по мережі до фактичного запиту. З Netflix Eureka [4] кожен клієнт може одночасно виступати в ролі сервера, щоб дублікувати свій статус на підключений вузол. Іншими словами, клієнт отримує список всіх підключених записів в реєстрі

служб і робить все подальші запити до будь-яким іншим службам за допомогою алгоритму балансування навантаження. Щоб отримати інформацію про присутність клієнта, він повинен відправити в системний сигнал серцебиття.

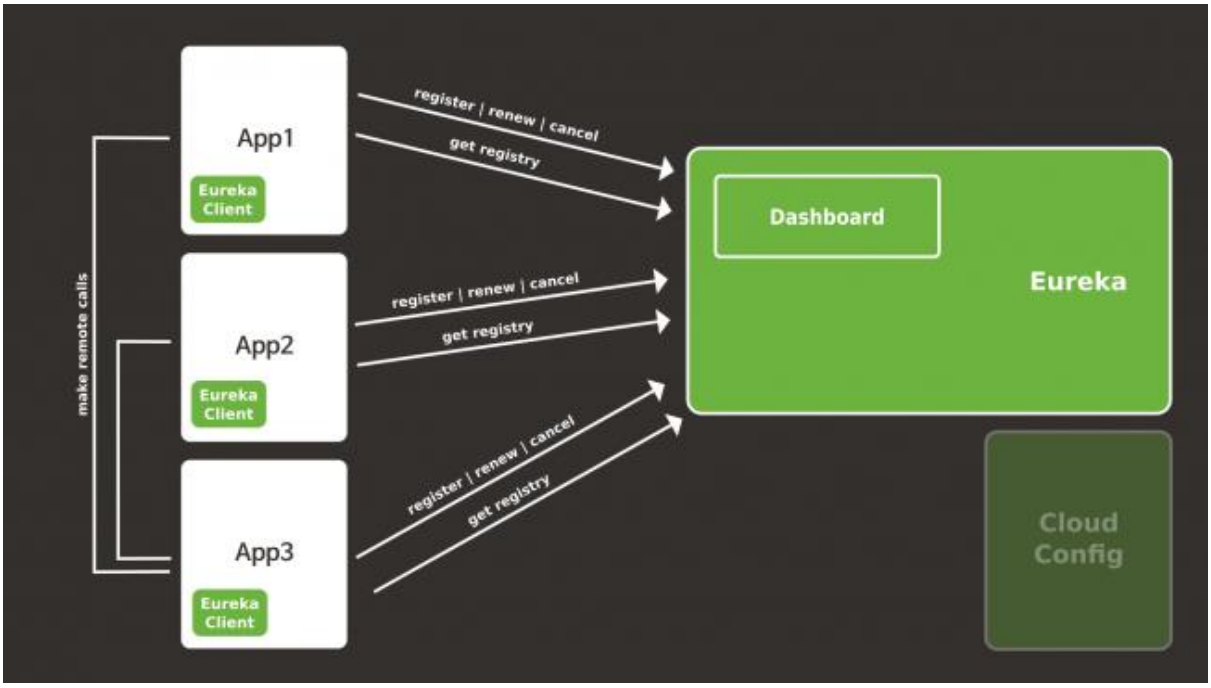

Рис. 2.3. Інфраструктура з використанням Spring Cloud Eureka

Клієнти Eureka отримують інформацію реєстру з сервера і кешують її локально. Після цього клієнти використовують цю інформацію для пошуку інших послуг. Ця інформація оновлюється періодично (кожні 30 секунд), отримуючи дельта-оновлення між останнім циклом вибірки і поточним. Дельта-інформація зберігається довше (близько 3 хвилин) на сервері, тому дельта-вибірки можуть знову повертати ті ж екземпляри. Клієнт Eureka автоматично обробляє дублюючу інформацію.

Отримавши дельти, клієнт Eureka звіряє інформацію з сервером, порівнюючи кількість примірників, що повертаються сервером, і, якщо інформація з якої-небудь причини не збігається, вся інформація реєстру витягується знову. Сервер Eureka кешує стисле корисне навантаження дельт, всього реєстру, а також кожної програми, а також не зтиснутою інформацію про них. Корисне навантаження також підтримує обидва формати JSON / XML. Клієнт Eureka

отримує інформацію в стислому форматі JSON, використовуючи клієнт Jersey Apache.

2.1.3. Open Feign

Feign є декларативним клієнтом веб-сервісу. Це полегшує написання клієнтів веб-сервісів. Для використання Feign створіть інтерфейс і анотує його. Він має підтримку підключаються анотацій, включаючи анотації Feign і анотації JAX-RS. Feign також підтримує підключаються кодери і декодери. В Spring Cloud додана підтримка анотацій Spring MVC і використання тих же HttpMessageConverters, які використовуються за замовчуванням в Spring Web. Spring Cloud інтегрує Ribbon і Eureka для забезпечення http-клієнта з балансуванням навантаження при використанні Feign.

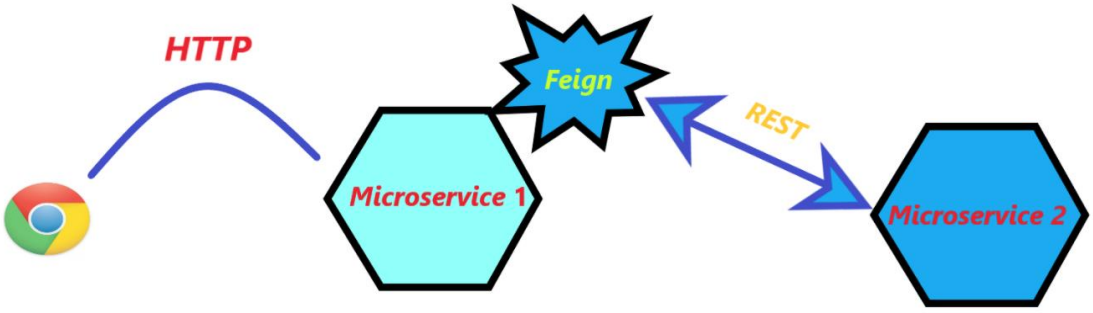

Рис. 2.4. Використання Open Feign [7]

2.1.4. Spring Cloud Config

Spring Cloud Config надає підтримку на стороні сервера і на стороні для зовнішньої конфігурації в розподіленої системі. З Config сервером у вас є центральне місце для керування зовнішніми властивостями для додатків у всіх середовищах. Концепції і клієнта, і сервера відповідають ідентичним абстракцій Spring Environment і PropertySource, тому вони дуже добре підходять для додатків Spring, але можуть використовуватися з будь-яким додатком, що працює на будь-якій мові. У міру того, як додаток переміщається по конвеєру розгортання від dev до тестування і в продакшн, ви можете управляти конфігурацією між цими середовищами і бути впевненим, що у

додатків є все, що їм потрібно для запуску при міграції. Реалізація серверної частини сховища за замовчуванням використовує git, тому вона легко підтримує марковані версії конфігураційних середовищ, а також доступна для широкого спектра інструментів для управління контентом. Легко додати альтернативні реалізації і підключити їх з конфігурацією Spring.

Ідея сервера конфігурації виникла з маніфесту 12-факторного додатки, пов'язаного з рекомендаціями по розробці кращих сучасних хмарних додатків. Він пропонує виводити властивості або файли ресурсів з сервера, де значення цих ресурсів змінюються під час виконання - зазвичай це різні конфігурації, які будуть відрізнятися в кожному середовищі.

Наприклад, припустимо, що одна служба залежить від іншої служби (викликається для певних бізнес-сценаріїв), і якщо URL-адресу залежною служби був змінений на щось інше. Потім зазвичай нам потрібно створити і розгорнути наш сервіс з оновленим URL. Тепер, якщо ми використовуємо 12-факторний підхід до додатків і якщо ми читаємо ці властивості конфігурації з зовнішньої служби, то нам просто потрібно оновити URL на сервері конфігурації і оновити цю конфігурацію служби клієнта, щоб використовувати оновлений URL.

2.1.5. Kafka

Apache Kafka [6] використовується для забезпечення зв'язку між виробниками і споживачами з використанням тим, заснованих на повідомленнях. Apache Kafka - це швидка, що масштабується, відмовостійка система обміну повідомленнями з підпискою. По суті, це платформа для високопродуктивних розподілених додатків нового покоління. Однією з кращих функцій Kafka є висока доступність, стійкість до збоїв вузлів і підтримка автоматичного відновлення. Ця функція робить Apache Kafka ідеальним для зв'язку і

інтеграції між компонентами великомасштабних систем даних в реальних системах даних.

Більш того, ця технологія заміняє звичайних брокерів повідомлень, здатних забезпечити більш високу пропускну здатність, надійність і реплікацію, таких як JMS, AMQP [22] і багато інших. Крім того, основна абстракція Kafka пропонує брокера Kafka, виробника Kafka і споживача Kafka. Брокер Kafka - це вузол в кластері Kafka, він використовується для збереження і реплікації даних. Виробник Kafka поміщає повідомлення в контейнер повідомлень, званий темою Kafka. Беручи до уваги, що Споживач Кафки витягує повідомлення з Теми Кафки. Коли ми передаємо дані з однієї програми до іншої, ми використовуємо систему обміну повідомленнями. Це призводить до того, що, не турбуючись про те, як обмінюватися даними, додатки можуть зосередитися тільки на даних. На концепції надійної черги повідомлень грунтується розподілений обмін повідомленнями. Хоча повідомлення асинхронно поміщаються в чергу між клієнтськими додатками і системою обміну повідомленнями. Існує два типи шаблонів обміну повідомленнями: система обміну повідомленнями типу «точка-точка» і «публікація-підписка». Тим не менше, більшість шаблонів повідомлень йдуть pub-sub.

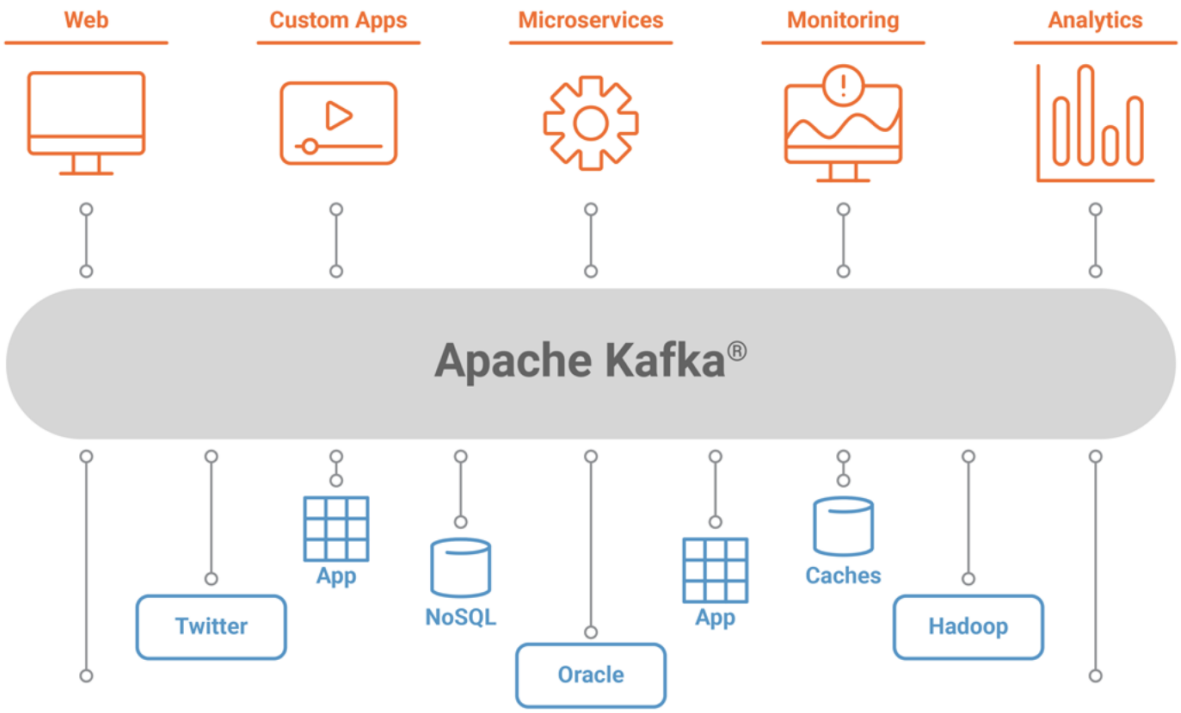

Рис. 2.5. Середовище використання Apache Kafka [6]

2.1.6. MySQL DB

MySQL [3] - це підтримувана Oracle система керування базами даних (СКБД) з відкритим вихідним кодом, заснована на мові структурованих запитів (SQL). MySQL працює практично на всіх платформах, включаючи Linux, UNIX і Windows. Хоча MySQL може використовуватися в широкому спектрі додатків, він найчастіше асоціюється з веб-додатками і онлайн-публікаціями.

MySQL - важливий компонент корпоративного стека з відкритим вихідним кодом під назвою LAMP. LAMP - це платформа веб-розробки, яка використовує Linux в якості операційної системи, Apache в якості веб-сервера, MySQL в якості системи керування базами даних і PHP в якості об'єктно-орієнтованої мови сценаріїв. (Іноді замість PHP використовується Perl або Python.)

Спочатку задуманий шведською компанією MySQL AB, MySQL був придбаний Sun Microsystems в 2008 році, а потім Oracle, коли він купив Sun в

2010 році. Розробники можуть використовувати MySQL відповідно до Стандартної громадської ліцензією GNU (GPL), але підприємства повинні отримати комерційну ліцензію від Oracle.

Сьогодні MySQL - це СУБД, що лежить в основі багатьох провідних веб-сайтів у світі і незліченних корпоративних і орієнтованих на споживачів веб-додатків, включаючи Facebook, Twitter і YouTube.

MySQL заснований на моделі клієнт-сервер. Ядром MySQL є сервер MySQL, який обробляє всі інструкції (або команди) бази даних. Сервер MySQL доступний як окрема програма для використання в мережевому середовищі клієнт-сервер і як бібліотека, яку можна вбудувати (або зв'язати) в окремі додатки.

MySQL працює разом з декількома службовими програмами, які підтримують адміністрування баз даних MySQL. Команди вирушають на MySQLServer через клієнт MySQL, встановлений на комп'ютері.

MySQL з самого початку був розроблений для швидкої обробки великих баз даних. Хоча MySQL зазвичай встановлюється тільки на одному комп'ютері, він може відправляти базу даних в кілька місць, оскільки користувачі можуть отримати до неї доступ через різні клієнтські інтерфейси MySQL. Ці інтерфейси відправляють оператори SQL на сервер, а потім відображають результати.

Переваги використання MySQL:

- Масштабованість і гнучкість. Сервер бази даних MySQL забезпечує максимальну масштабованість, володіючи здатністю обробляти глибоко вбудовані додатки, що становлять лише 1 МБ, для роботи величезних сховищ даних, що містять терабайти інформації. Гнучкість платформи - невід'ємна частина MySQL, при цьому підтримуються всі різновиди Linux, UNIX і Windows. І, звичайно ж, природа

MySQL з відкритим вихідним кодом дозволяє виконувати повну настройку для тих, хто хоче додати унікальні вимоги до сервера бази даних.

- Висока продуктивність. Унікальна архітектура механізму зберігання дозволяє фахівцям по базах даних налаштовувати сервер бази даних MySQL спеціально для конкретних додатків, в результаті чого досягаються приголомшливі показники продуктивності. Незалежно від того, чи є передбачуване додаток системою високошвидкісної обробки транзакцій або веб-сайтом великого обсягу, обслуговуючим мільярд запитів в день, MySQL може задовольнити найвищі вимоги до продуктивності будь-якої системи. Завдяки високошвидкісним утилітам завантаження, відмітною кешах пам'яті, повнотекстових індексах і іншим механізмам підвищення продуктивності MySQL пропонує всі необхідні засоби для сьогоднішніх критично важливих бізнес-систем.

- Висока доступність. Бездоганна надійність і постійна доступність є відмінними рисами MySQL, оскільки клієнти покладаються на MySQL, щоб гарантувати цілодобову працездатність. MySQL пропонує безліч варіантів високої доступності від конфігурацій високошвидкісний реплікації головний / підлеглий, до спеціалізованих серверів кластерів, що забезпечують миттєве перемикання при відмові, до сторонніх

постачальників, що пропонують унікальні рішення високої доступності для сервера баз даних MySQL.

- Надійна підтримка транзакцій. MySQL пропонує один з найпотужніших механізмів транзакційних баз даних на ринку. Можливості включають повну підтримку транзакцій ACID (атомарному, узгоджену, ізольовану, надійну), необмежену блокування на рівні рядків, можливість розподілених транзакцій і підтримку транзакцій з декількома версіями, при яких зчитувачі ніколи не блокують записи, і навпаки. Повна цілісність даних також забезпечується за рахунок примусової посилальної цілісності сервера, спеціалізованих рівнів ізоляції транзакцій і миттєвого виявлення взаімоблокіровок.

- Сильні сторони Інтернету і сховищ даних. MySQL де-факто є стандартом для веб-сайтів з високою відвідуваністю завдяки високопродуктивної механізму запитів, надзвичайно швидкої можливості вставки даних і сильну підтримку спеціалізованих веб-функцій, таких як швидкий повнотекстовий пошук. Ці ж сильні сторони можна застосувати й до середах сховищ даних, де MySQL масштабується до терабайтного діапазону або для окремих серверів, або для горизонтально масштабованих архітектур. Інші функції, такі як таблиці основний пам'яті, B-дерево і хеш-індекси, а також стислі архівні таблиці, які знижують вимоги до сховища до вісімдесяти

відсотків, роблять MySQL сильним відмінністю як для веб-додатків, так і для додатків бізнес-аналітики.

- Надійний захист даних. Оскільки захист інформаційних активів корпорацій - це робота номер один для професіоналів в області баз даних, MySQL пропонує виняткові функції безпеки, що забезпечують абсолютний захист даних. Що стосується аутентифікації бази даних, MySQL надає потужні механізми, що гарантують, що тільки авторизовані користувачі мають доступ до сервера бази даних, з можливістю блокувати користувачів аж до рівня клієнтської машини. Також надається підтримка SSH і SSL для забезпечення безпечних і надійних з'єднань. Передбачена деталізована структура привілеїв об'єктів, так що користувачі бачать лише ті дані, які їм слід, а потужні функції шифрування і дешифрування даних гарантують, що конфіденційні дані захищені від несанкціонованого перегляду. Нарешті, утиліти резервного копіювання та відновлення, що надаються MySQL і сторонніми постачальниками програмного забезпечення, дозволяють виконувати повне логічне і фізичне резервне копіювання, а також повне відновлення і відновлення на певний момент часу.

- Комплексна розробка додатків. Одна з причин, по якій MySQL є найпопулярнішою в світі базою даних з відкритим вихідним кодом, полягає в тому, що вона забезпечує всебічну підтримку для будь-яких потреб

розробки додатків. У базі даних можна знайти підтримку збережених процедур, тригерів, функцій, уявлень, курсорів, стандартного SQL SQL і багато чого іншого. Для вбудованих додатків доступні спільні бібліотеки, що дозволяють вбудувати підтримку бази даних MySQL практично в будь-який додаток. MySQL також надає з'єднувачі і драйвери (ODBC, JDBC і т. Д.), які дозволяють всім формам додатків використовувати MySQL в якості кращого сервера управління даними. Неважливо, PHP це, Perl, Java, Visual Basic або .NET, MySQL пропонує розробникам додатків все, що їм потрібно для успішної побудови інформаційних систем, керованих базами даних.

- Легкість керування. MySQL пропонує виняткову можливість швидкого запуску, при цьому середній час від завантаження програмного забезпечення до завершення установки становить менше п'ятнадцяти хвилин. Це правило виконується незалежно від того, чи є платформа Microsoft Windows, Linux, Macintosh або UNIX. Після установки функції самоврядування, такі як автоматичне розширення простору, автоматичний перезапуск і зміна динамічної конфігурації, знімають більшу частину навантаження з вже перевантажених роботою адміністраторів баз даних. MySQL також надає повний набір графічних інструментів управління та міграції, які дозволяють адміністраторам баз даних управляти, усувати

неполадки і контролювати роботу багатьох серверів MySQL з однієї робочої станції. Для MySQL також доступні багато інструментів сторонніх постачальників програмного забезпечення, які обробляють різні завдання, від проектування даних і ETL до повного адміністрування бази даних, управління завданнями і моніторингу продуктивності.

- Свобода відкритого вихідного коду і цілодобова підтримка. Багато корпорацій не наважуються повністю присвятити себе використання програмного забезпечення з відкритим вихідним кодом, тому що вони вважають, що не можуть отримати той тип підтримки або мережі безпеки професійних послуг, на які вони в даний час покладаються з пропрієтарним програмним забезпеченням для забезпечення загального успіху своїх ключових додатків. Часто виникають питання про компенсацію. Ці проблеми можуть бути вирішені за допомогою MySQL, оскільки повна цілодобова підтримка та компенсація доступні через MySQL Enterprise. MySQL не є типовим проектом з відкритим вихідним кодом, оскільки все програмне забезпечення належить і підтримується Oracle, і з цієї причини доступна унікальна модель вартості і підтримки, яка забезпечує унікальне поєднання свободи відкритого вихідного коду і надійного програмного забезпечення з підтримкою.

- Найнижча сукупна вартість володіння. Шляхом міграції поточних додатків для роботи з базами даних в MySQL або використання MySQL для нових проектів розробки корпорації досягають економії, яка в багато разів обчислюється семизначними цифрами. Завдяки використанню сервера бази даних MySQL і масштабованих архітектур, що використовують недороге стандартне обладнання, корпорації виявляють, що вони можуть досягти приголомшливих рівнів масштабованості і продуктивності при набагато менших витратах, ніж у пропрієтарних і постачальники програмного забезпечення для масштабування. Крім того, надійність і простота обслуговування MySQL означає, що адміністратори баз даних не витрачають час на усунення неполадок, пов'язаних з продуктивністю або простоями, а замість цього можуть зосередитися на наданні позитивного впливу на завдання більш високого рівня, пов'язані з бізнес-стороною даних.

2.1.7. Docker

Docker - це платформа контейнеризації, яка використовується для упаковки вашої програми і всіх його залежностей разом у вигляді контейнерів, щоб ваше додаток працювало без проблем в будь-якому середовищі, яка може бути розробкою, тестуванням або виробництвом. Docker - це інструмент, призначений для спрощення створення, розгортання і запуску додатків з використанням контейнерів.

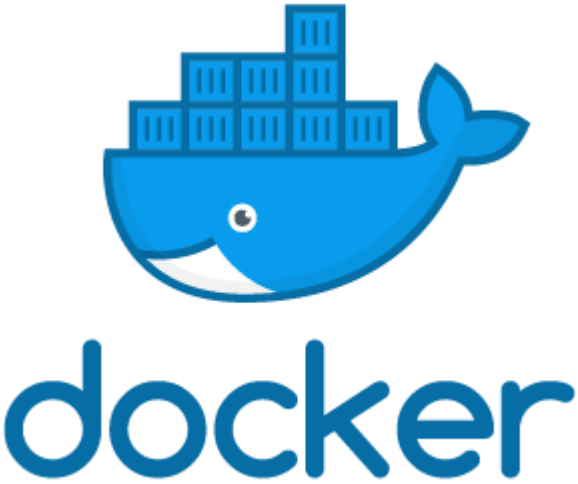

Рис. 2.6. Логотип Docker [8]

Docker - це провідна в світі платформа для створення програмних контейнерів. Він був запущений в 2013 році компанією Dotcloud. Він написаний на мові Go. З моменту запуску Docker пройшло всього шість років, але співтовариства вже перейшли на нього з віртуальних машин. Docker призначений для використання як розробниками, так і системними адміністраторами, що робить його частиною багатьох наборів інструментів Devops. Розробники можуть писати код, не турбуючись про тестової та виробниче середовище. Системним адміністраторам не потрібно турбуватися про інфраструктуру, оскільки Docker може легко збільшувати і зменшувати кількість систем. Докер вступає в гру на етапі розгортання циклу розробки програмного забезпечення.

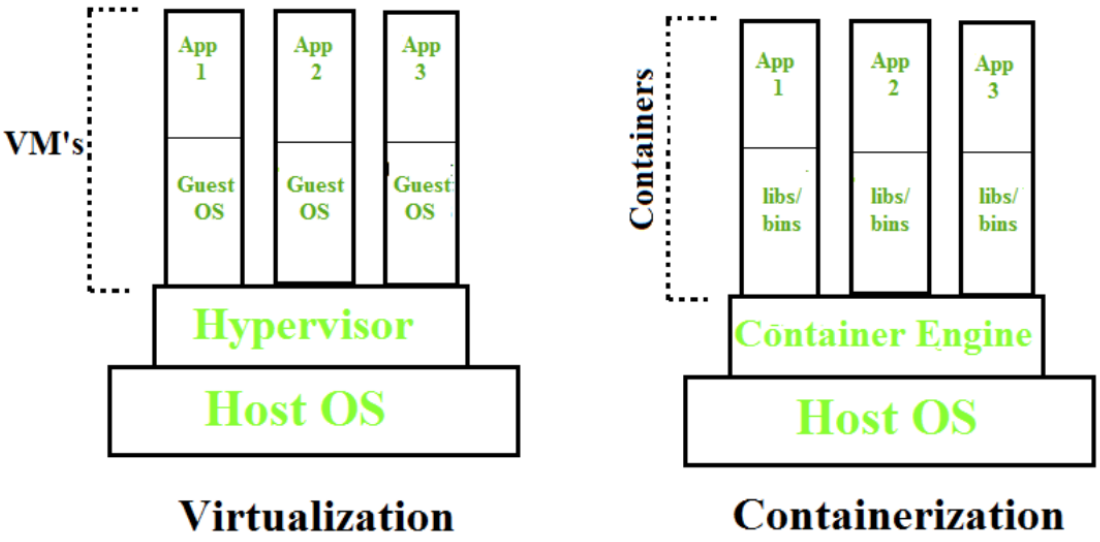

Рис. 2.7. Різниця між докер контейниризацією та віртуалізацією

Контейнеризація - це віртуалізація на основі ОС, яка створює кілька віртуальних одиниць в просторі користувача, відомих як контейнери. Контейнери використовують один і той же ядро хоста, але ізольовані один від одного за допомогою приватних просторів імен і механізмів управління ресурсами на рівні ОС. Віртуалізація на основі контейнерів забезпечує інший рівень абстракції з точки зору віртуалізації і ізоляції в порівнянні з гіпервізорами. Гіпервізор використовують багато обладнання, що призводить до накладних витрат з точки зору віртуалізації обладнання та драйверів віртуальних пристроїв. Повна операційна система (наприклад, Linux, Windows) працює поверх цього віртуалізованого обладнання в кожному примірнику віртуальної машини.

Але контейнери, навпаки, реалізують ізоляцію процесів на рівні операційної системи, що дозволяє уникнути таких накладних витрат. Ці контейнери працюють поверх того ж ядра загальної операційної системи, що і базова хост-машина, і в кожному контейнері може виконуватися один або кілька процесів. У контейнерах вам не потрібно попередньо виділяти оперативну пам'ять, вона виділяється динамічно під час створення контейнерів, в той час як в віртуальних машинах вам потрібно спочатку

попередньо виділити пам'ять, а потім створити віртуальну машину. Контейнерна обробка забезпечує краще використання ресурсів у порівнянні з віртуальними машинами і короткий процес завантаження. Це наступна еволюція віртуалізації.

Контейнери можуть працювати практично де завгодно, що значно спрощує розробку і розгортання: в операційних системах Linux, Windows і Mac; на віртуальних машинах або на «голому залізі», на машині розробника або в локальних центрах обробки даних; і, звичайно ж, в публічному хмарі. Контейнери віртуалізують ЦП, пам'ять, сховище і мережеві ресурси на рівні ОС, надаючи розробникам ізольоване уявлення ОС, логічно ізольоване від інших додатків. Docker - це найпопулярніший з доступних форматів контейнерів з відкритим вихідним кодом, який підтримується в Google Cloud Platform і Google Kubernetes Engine.

Docker став популярним в даний час з-за переваг, що надаються контейнерами Docker.

Основними перевагами Docker є:

- Швидкість - швидкість контейнерів Docker в порівнянні з віртуальною машиною дуже висока. Час, необхідний для створення контейнера, дуже короткий, тому що вони дійсно маленькі і легкі. Розробка, тестування та розгортання можуть бути виконані швидше, оскільки контейнери невеликі. Контейнери можуть бути відправлені на тестування після їх створення, а потім у виробничу середу.
- Переносимість - додатки, вбудовані в контейнери докерів, надзвичайно портативні. Ці портативні програми можна легко переміщати куди завгодно як єдиний елемент, при цьому їх продуктивність залишається колишньою.
- Масштабованість - Docker може бути розгорнутий на декількох фізичних серверах, серверах даних і хмарних платформах. Його

також можна запустити на будь-якій машині Linux. Контейнери можна легко перенести з хмарної середовища на локальний хост, а звідти назад в хмару в швидкому темпі.

- Щільність - Docker використовує доступні ресурси більш ефективно, оскільки не використовує гіпервізор. Це причина того, що на одному хості може працювати більше контейнерів, ніж на віртуальних машинах. Контейнери Docker мають більш високу продуктивність через їх високої щільності і відсутності непродуктивних витрат ресурсів.

2.1.8. Spring Data JPA

JPA - це специфікація, яка визначає API для об'єктно-реляційних зіставлень і для управління постійними об'єктами. Hibernate [18] і EclipseLink [21] - дві популярні реалізації цієї специфікації.

Spring Data JPA додає шар поверх JPA. Це означає, що він використовує всі функції, визначені специфікацією JPA, особливо зіставлення сутностей і асоціацій, управління життєвим циклом сутностей і можливості запитів JPA. Додатково до цього Spring Data JPA додає свої власні функції, такі як реалізація шаблону сховища без коду і створення запитів до бази даних з імен методів.

Переваги використання Spring Data JPA:

- Паттерн репозиторію. Шаблон сховища - один з найпопулярніших шаблонів, пов'язаних з постійністю. Він приховує конкретні деталі реалізації сховища даних і дозволяє реалізувати бізнес-код на більш високому рівні абстракції. Реалізувати цей шаблон не так вже й складно, але написання стандартних операцій CRUD для кожної сутності створює багато повторюваного коду. Spring Data JPA надає вам набір інтерфейсів сховища, який вам потрібно тільки розширити, щоб визначити конкретний репозиторій для однієї з ваших сутностей.

- Зменшення коду. Щоб зробити це ще простіше, Spring Data JPA надає реалізацію за замовчуванням для кожного методу, визначеного одним з інтерфейсів його сховища. Це означає, що вам більше не потрібно виконувати базові операції читання або запису. І навіть в цьому випадку для всіх цих операцій не потрібно багато коду, відсутність необхідності їх реалізації робить життя трохи простіше і знижує ризик появи дурних помилок.
- Генеровані запити. Ще одна зручна функція Spring Data JPA - створення запитів до бази даних на основі імен методів. Якщо ваш запит не надто складний, вам просто потрібно визначити метод в інтерфейсі сховища з ім'ям, яке починається з findBy. Потім Spring аналізує ім'я методу і створює для нього запит. Усередині Spring генерує запит JPQL на основі імені методу, встановлює надані параметри методу як значення параметрів прив'язки, виконує запит і повертає результат.

2.2. Архітектура системи – взаємодія усіх модулів системи

Для реалізації додатку я використав мікросервісну архітектуру.
Мікросервісна архітектура - архітектурний підхід до розробки програмних компонентів за рахунок розбивання їх на атомарні, ізольовані сервіси котрі взаємодіють один з одним через певні канали передачі даних.

Коротше кажучи, архітектурний стиль мікросервіса - це підхід до розробки окремих додатків у вигляді набору невеликих сервісів, кожен з яких працює в своєму власному процесі і взаємодіє із полегшеними механізмами часто API-інтерфейсом HTTP-ресурсів. Ці сервіси побудовані навколо бізнес-можливостей. Існує мінімальний рівень централізованого управління цими службами.

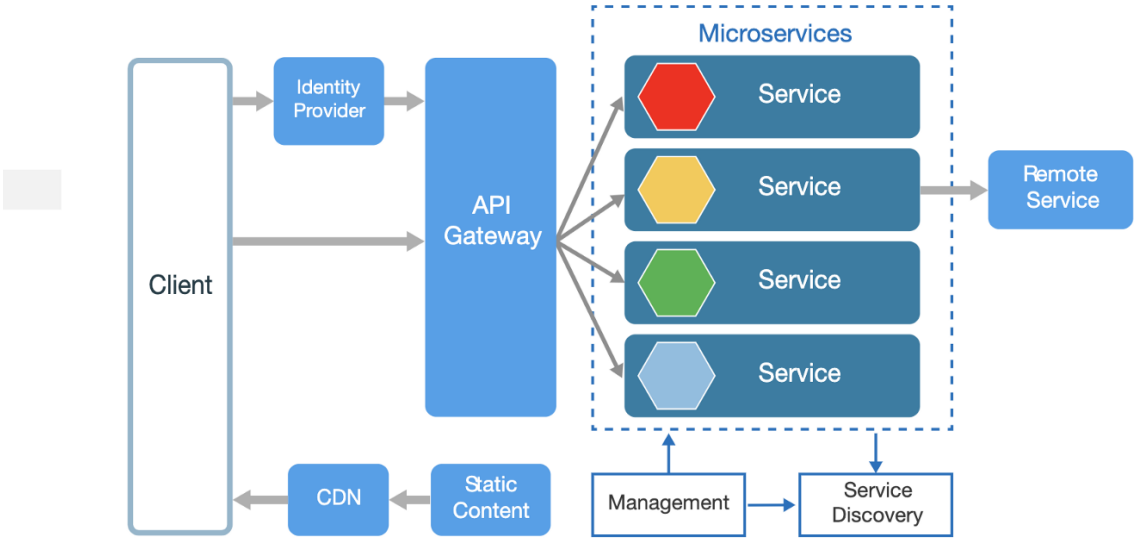

Рис. 2.8. Мікросервісна архітектура [17]

Я не можу сказати, що існує формальне визначення архітектурного стилю мікросервісів, але ми можемо спробувати описати те, що ми бачимо як загальні характеристики для архітектур, які відповідають мітці. Як і в разі будь-якого визначення, яке описує загальні характеристики, не всі мікросервісние архітектури володіють всіма характеристиками, але ми очікуємо, що більшість мікросервісних архітектур мають більшістю характеристик.

Там, де необхідно розділити великі додатки на частини, часто управління фокусується на технологічному рівні, що призводить до командам призначеного для користувача інтерфейсу, командам логіки на стороні сервера і командам баз даних. Коли команди розділені за цими напрямками, навіть прості зміни можуть привести до міжгрупового проекту, який вимагає часу і схвалення бюджету. Розумна команда буде оптимізувати цю ситуацію і використовувати менше з двох зол - просто нав'язати логіку того додатком, до якого у них є доступ. Логіка всюди іншими словами. Мікросервісний підхід до поділу відрізняється, розділяючись на послуги, організовані навколо можливостей бізнесу. Такі служби використовують широкосмугову реалізацію програмного забезпечення для цієї бізнес-сфери, включаючи користувальницький інтерфейс, постійне сховище і будь-які зовнішні

взаємодії. Отже, команди є крос-функціональними, включаючи весь спектр навичок, необхідних для розробки: призначений для користувача досвід, бази даних і управління проектами.

Перелік найкращих практик використання мікросервісної архітектури:

- Окремі мікросервіси на основі бізнес-доменів. Для банківського додатка це може означати розбиття основної банківської системи, повідомлень, платежів за рахунками, кредитів і т.д. на окремі мікросервіси.
- Перенесіть наскрізні проблеми, такі як аутентифікація і завершення SSL, на шлюз. Amazon [19] і Azure [20] обидва мають функції шлюзу API
- Асинхронний. Потрібно структуризовувати свої взаємозалежності мікросервісов таким чином, щоб виклики API не збільшували час відгуку.
- Керований подіями - як і вище, бізнес-вимоги, такі як повідомлення, завдання зі збору пам'яті, повинні запускатися з подіями і виконуватися у фоновому режимі.

Висновок до другого розділу

Результатом аналізу доступних технологій можем обрати об’єктно орієнтовану мову програмування Java та фреймворк Spring для реалізації шаблону Dependency Injection.

Для того щоб побудувати мікросервісну інфраструктуру я буду використовувати проект Spring Cloud з його можливостями:

- Реєстрація сервісів
- Балансування навантаження
- Розподілена конфігурація

Кожен мікросервіс буде Spring Boot додатком з основним скоупом функціоналу фреймворку Spring.

Для збереження даних, була обрана реляційна база даних MySQL через її швидкодію, зручність та високий поріг входження.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ СИСТЕМИ

3.1. Контекстна діаграма

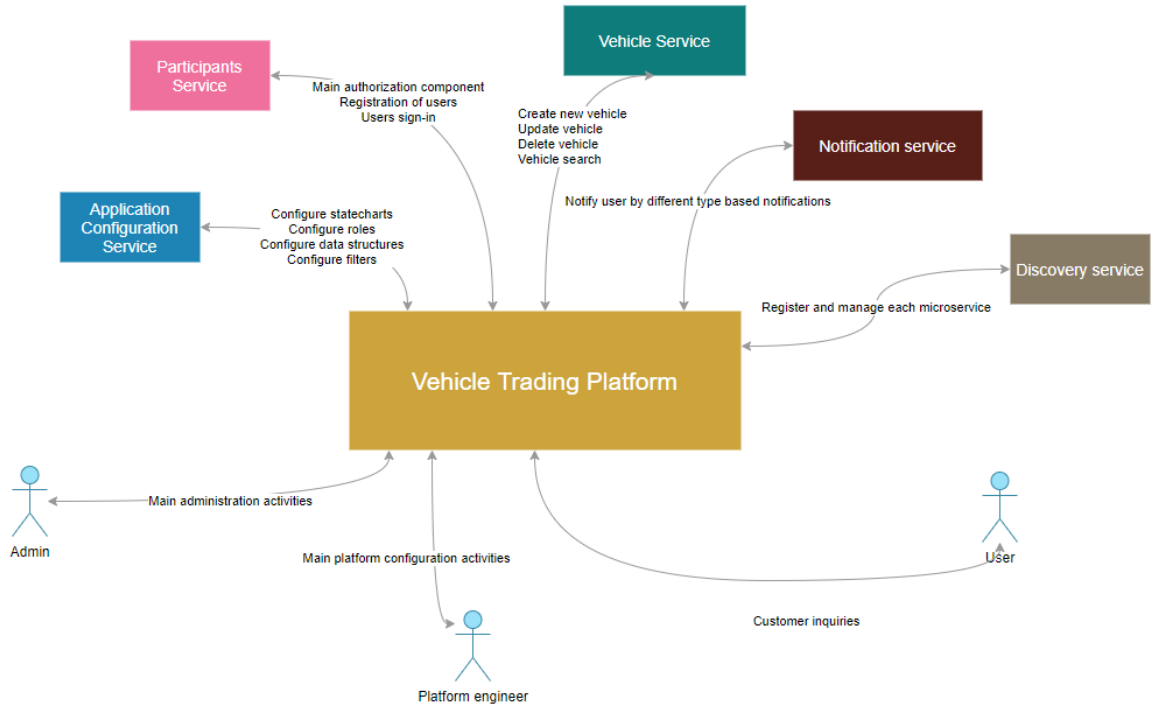

Рис. 3.1. C4 діаграма основних компонентів системи

Здійснимо опис таблиць:

Таблиця 3.1

Опис таблиць бази даних

Ім'я	**Детальний опис**
account	Таблиця зберігає зареєстрованих користувачів, поле password закодовано за допомогою алгоритму BCrypt. Також зберігається спеціальний прапор (enabled) котрий показує включений користувач чи ні.
role	Таблиця зберігає імена ролей для кожного користувача. Усі ролі повинні починатися з префіксу ROLE_.

Продовження таблиці 3.1

Ім’я	**Детальний опис**
account_role	Таблиця зберігає зв’язки між користувачем та роллю (відношення many-to-many).
verification_token	Таблиця що зберігає токени для перевірки того чи зареєстрований користувач підтвердив реєстрацію.
password_reset_token	Таблиця що зберігає токени для підтвердження зміни паролю користувачем.
car	Таблиця що зберігає мета інформацію по автомобілю, такі як: основні дані, рік, ціна, опис, url картинки, прапор чи дане оголошення знаходиться у основному потоці оголошень (модерація виконується адміністратором).

3.2. Об’єктне представлення таблиць за допомогою ORM

Опис моделей (сутностей) даних.

Таблиця 3.2

Об’єктне представлення моделей бази даних

Ім’я сутності	**Список полів**	**Опис сутності**
AccountEntity	• id – ідентифікатор • login – ім’я користувача • password – пароль • roles – список ролей	Зберігає основну мета інформацію про користувача. Містить у собі список сутностей ролей.

Продовження таблиці 3.2

Ім’я сутності	**Список полів**	**Опис сутності**
PasswordResetTokenEntity	• id – ідентифікатор • token – токен у вигляді символьного рядку • accountEntity – сутність аккаунту, прив’язана за допомогою відношення один-до-одного • expiryDate – дата закінчення валідності токену	Зберігає у собі символьне представлення токену відновлення паролю а також його мета інформацію.
VerificationTokenEntity	• id – ідентифікатор • token – токен у вигляді символьного рядку • accountEntity – сутність аккаунту, прив’язана за допомогою відношення один-до-одного • expiryDate – дата закінчення токену	Зберігає у собі символьне представлення токену підтвердження аккаунту а також його мета інформацію.
RoleEntity	• id – ідентифікатор • name – ім’я ролі • description – детальний опис ролі	Зберігає у собі мета представлення ролі користувача.

Продовження таблиці 3.2

Ім’я сутності	**Список полів**	**Опис сутності**
CarEntity	• id – ідентифікатор • name – ім’я автомобілю • year – рік випуску • price – ціна автомобілю • enabled – прапор, що відповідає за те чи потрапив даний автомобіль до основної дошки оголошень • accountEntity – сутність аккаунту користувача, зв’язується за допомогою відношення багато-до-одного • description – детальний опис автомобілю • imageUrl – ссилка на завантажене зображення автомобілю	Зберігає основну мета-інформацію по автомобілю. Зв’язане з сутністю аккаунт щоб можна було робити пошук по автомобілів по конкретному користувачу.

Опис рівня репозиторіїв:

Таблиця 3.3

Опис репозиторіїв

Ім’я репозиторію	**Запити репозиторію до бд**
AccountRepository	• AccountEntity findByLogin(String login); – пошук екземпляру аккаунту по імені користувача
CarRepository	• List<CarEntity> findByPriceLessThanEqual(Integer price); - пошук списку машин по ціні менших ніж задана ціна • List<CarEntity> findByYearBetweenAndPriceBetween(Integer yearFrom, Integer yearTo, Integer priceFrom, Integer priceTo); - пошук списку машин по діапазону років та цін • List<CarEntity> findByAccountEntity(AccountEntity accountEntity); - пошук списку машин по сутності аккаунта • List<CarEntity> findNotActivatedCars(); - пошук списку автомобілів котрі ще не були додані в основну дошку оголошень • List<CarEntity> findActivatedCars(); - пошук списку автомобілів котрі вже були додані в основну дошку оголошень

Продовження таблиці 3.3

Ім’я репозиторію	**Запити репозиторію до бд**
(PasswordReset/Verification) TokenRepository	• TokenEntity findByToken(String token); - пошук екземпляру сутності токену по імені токену • TokenEntity findByAccountEntity(AccountEntity accountEntity); - пошук екземпляру сутності токену по сутності аккаунта • Stream<TokenEntity> findAllByExpiryDateLessThan(Date now); - пошук списку сутностей токену, що минули • void deleteByExpiryDateLessThan(Date now); - видалити усі токени що минули • void deleteAllExpiredSince(Date now); - видалити усі токени що минули з конкретної дати
RoleRepository	RoleEntity findByName(String name); - пошук сутності ролі по її імені

3.3. Опис запитів до API

Запити для ресурсу “/user”.

Таблиця 3.4

Запити до API для керування користувачем

Тип	**Ім'я**	**Параметри**	**Дія**	**Доступ**

POST	/add	CreateAccountDto:{
		email,
		password
		}
			Реєстрація користувача	ALL

GET	/confirm	token – токен користувача	Підтвердити реєстрацію користувача	ALL

POST	/token/resend	EmailDto: {
		email
		}
			Повторна відправка посилання для підтвердження реєстрації	ALL

POST	/password/forgot	EmailDto: {
		email
		}
			Функція забутого паролю	ALL

GET	/password/reset	email – емейл користувача token - токен	Після переходження за посиланням відновлення паролю, відобразиться форма вводу нового паролю	ALL

Продовження таблиці 3.4

Тип	Ім’я	Параметри	Дія	Доступ
POST	/password/save	resetToken – токен відновлення паролю password – новий пароль	Оновити існуючий пароль на новий	ALL

Запити для ресурсу “/api/car”.

Таблиця 3.5

Запити до API для керування оголошеннями

Тип	Ім’я	Параметри	Дія	Доступ
POST	/add	CreateCarDto: { name, year, price, description }	Створити новий автомобіль	USER, ADMIN
GET	/cars	yearFrom – рік після, yearTo – рік до, priceFrom – ціна після, priceTo – ціна до	Повертає список автомобілів згідно фільтрації	USER, ADMIN
GET	/activate	id – ідентифікатор автомобіля	Переносить автомобіль в основну дошку оголошень	ADMIN

Продовження таблиці 3.5

Тип	**Ім'я**	**Параметри**	**Дія**	**Доступ**
GET	/cars/user	-	Отримати список власних оголошень автомобілів користувача	USER, ADMIN
GET	/cars/user/{login}	login – ім'я користувача	Отримати список автомобілів по імені користувача	USER, ADMIN
GET	/cars/disabled	-	Отримати список оголошень автомобілів що не були додані в основну дошку оголошень	ADMIN
DELETE	/delete	id – ідентифікатор автомобіля	Видалити автомобіль по його ідентифікатору	USER, ADMIN
GET	/files/{filename:.+}	filename – ім'я файлу	Отримати файл по його імені	USER, ADMIN

Продовження таблиці 3.5

Тип	Ім’я	Параметри	Дія	Доступ
POST	/upload/{carId}/image	carId – ідентифікатор автомобіля file - файл	Завантажити файл на сервер	USER, ADMIN

3.4. Опис основних компонентів системи

3.4.1. Налаштування платформи

Налаштування специфікацій потенційних користувачів, структур даних та ролей. Ми можемо зазначити які будуть доступні ролі у нашій платформі. Кожна роль має свій набір необхідних атрибутів. Наприклад: ім’я ролі, унікальний ідентифікатор, механізм приєднання до ролі, приналежність ролі до сімейства адмініструючих і т. д. Приєднаний у мережу користувач відображає приєднання до конкретної ролі (ролей).

Конфігурація основних флоу трейдингу технікою – користувач налаштовує які саме будуть існувати статуси оголошень у системі, механізми переходів та оновлень цих статусів.

Кофігурація структур даних – користувач має змогу надати динамічну конфігурацію структур оголошень. За допомогою використання документоорінтованих баз даних (такої як MongoDB), я можу забезпечити динамічне налаштування потенційних моделей даних. Ці моделі підтримують версії та ревізії задля забезпечення надійної оберненої сумісності.

3.4.2. Функціонал користувача налаштованої платформи

Перегляд оголошень – користувач може переглядати, шукати, фільтрувати необхідний набір оголошень та здійснювати потрібні агрегації.

Менеджмент оголошень – користувач може створювати/додавати/видаляти оголошення. Усі переходи статусів оголошень базуються на конфігурації станів оголошень.
Менеджмент аккаунтів – користувач має змогу верифікувати свою персону за допомогою різноманітних каналів зв’язку (смс, імейл і т.д.). Також має змогу відновити втрачений пароль, оновити пароль, змінити основні налаштування профілю.

3.5. Роутинг запитів

Поширеною проблемою при створенні мікросервісов є надання єдиного інтерфейсу для споживачів вашої системи. Той факт, що ваші сервіси розділені на невеликі складаються додатки, не повинен бути видимий для користувачів або приводити до значних зусиль при розробці.

Щоб вирішити цю проблему, Netflix (великий прихильник мікросервісов) створив проксі-сервер Zuul з відкритим вихідним кодом. Zuul [23] [24] - це прикордонний сервіс, який передає запити до декількох допоміжних сервісів. Він забезпечує єдину «вхідні двері» в вашу систему, яка дозволяє браузеру, мобільному додатку або іншому призначеному для користувача інтерфейсу використовувати сервіси з декількох хостів без управління спільним використанням ресурсів між джерелами (CORS) і аутентифікації для кожного з них. Я використовую його для управління правилами маршрутизації, фільтрами і балансуванням навантаження у вашій системі.

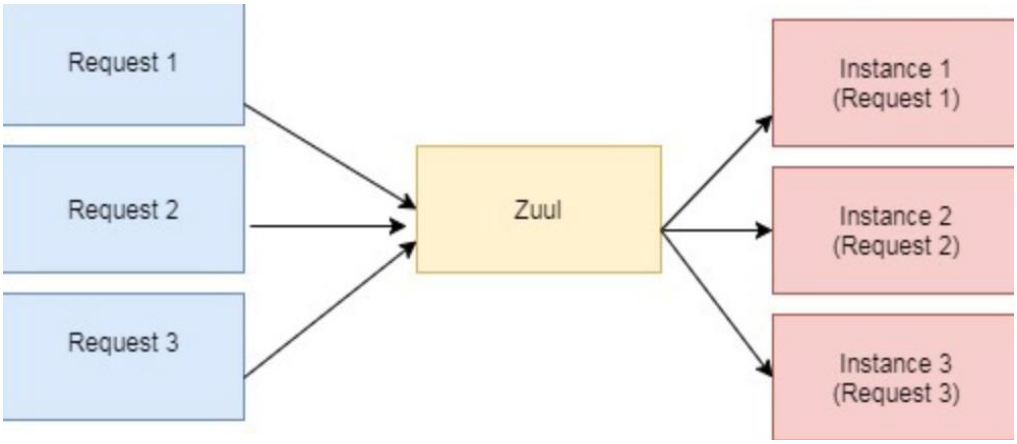

Рис. 3.2. Схема балансування навантаження у моїй системі

Розгорнутий мною сервер Zuul [23] поставляється з попередньо упакованої стрічкою, балансувальник навантаження на стороні клієнта і Hystrix, бібліотекою відмовостійкості. Обидва проекти є частиною пакета Netflix OSS, що означає, що вони легко інтегруються з Zuul.

Налаштування серверу Zuul:

```
zuul:
  routes:
    httpbin:
      path: /**
      serviceId: httpbin
httpbin:
  ribbon:
    listOfServers: httpbin.org,eu.httpbin.org
ribbon:
  eureka:
    enabled: false
```

Згідно конфігурації, я вказую Zuul пересилати всі запити службі httpbin, яка визначається після запису zuul. Запис httpbin визначає доступні сервери: httpbin.org і його європейський аналог, eu.httpbin.org. Якщо перший хост виходить з ладу, проксі-сервер переключиться на другий хост. Він також відключає виявлення Eureka для стрічки.

Висновок до третього розділу

На основі реалізації системи, можемо виділити основні компоненти платформи:

- Applications configuration service
- Participants service
- Vehicle service
- Notification service
- Discovery service
- Load balancer

Компоненти, котрі являються мікросервісами мають свій власний persistency layer. У моїй системі можемо виділити 2 основних механізм обміну даними між мікросервісами: синхронний, асинхронний. Синхронний обмін буде забезпечуватись через HTTP протокол, за допомогою REST викликів. Асинхронний обмін буде реалізовано за допомогою брокерів повідомлень. Я обрав технологію обміну Kafka, так як вона забезпечує високу пропускну здатність та швидкість передачі даних.

Кожен сервіс реєструється у так званому реєстрі сервісів (Discovery service). Це дає змогу моніторити стан кожного із сервісів. Також реєстр забезпечує маппинг імен сервісів з їх беспосередніми фізичними ip адресами.

Також використав механізм клієнтського балансування навантаження, задля забезпечення відмовостійкості.

РОЗДІЛ 4

РЕЗУЛЬТАТИ ТА ТЕСТУВАННЯ ДОДАТКУ

4.1. Вибірки даних із баз даних мікросервісів

Table CSV

id	first_name	last_name	email	gender
1	Clint	Wolver	cwolver0@godaddy.com	Male
2	Ike	Balsillie	ibalsillie1@reference.com	Male
3	Stacee	Biagi	sbiagi2@sogou.com	Male
4	Winni	Verillo	wverillo3@rakuten.co.jp	Female
5	Kaylee	McCafferky	kmccafferky4@kickstarter.com	Female
6	Bax	Bode	bbode5@google.co.jp	Male
7	Elisabeth	Blanchard	eblanchard6@canalblog.com	Female
8	Katti	O'Dea	kodea7@senate.gov	Female
9	Joni	Remon	jremon8@xinhuanet.com	Female
10	Keenan	Mitchley	kmitchley9@skyrock.com	Male
11	Jordan	Dybald	jdybalda@oracle.com	Female
12	Berri	Scarfe	bscarfeb@dailymail.co.uk	Female
13	Arv	MacCambridge	amaccambridgec@cbsnews.com	Male
14	Brynne	Purcell	bpurcelld@scientificamerican.com	Female
15	Berk	Gavrieli	bgavrielie@cloudflare.com	Male
16	Goraud	Grzeszczyk	ggrzeszczykf@java.com	Male
17	Fionna	Gery	fgeryg@nymag.com	Female
18	Launce	Bittlestone	lbittlestoneh@ca.gov	Male
19	Powell	Carty	pcartyi@google.nl	Male
20	Lita	Dymoke	ldymokej@cbslocal.com	Female
21	Sebastiano	Seawell	sseawellk@blinklist.com	Male

Рис. 4.1. Дані таблиці користувачів

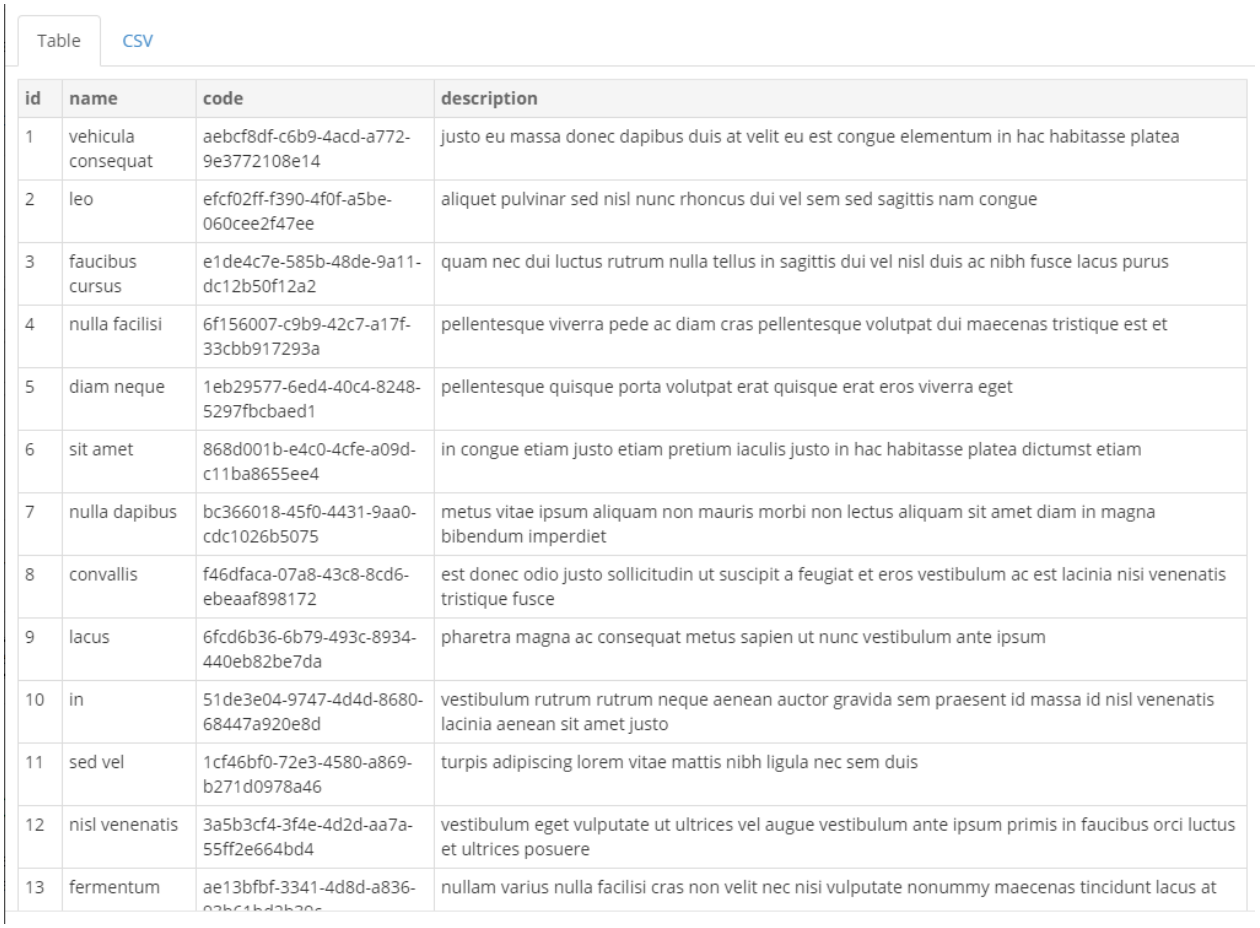

Table CSV

id	name	code	description
1	vehicula consequat	aebcf8df-c6b9-4acd-a772-9e3772108e14	justo eu massa donec dapibus duis at velit eu est congue elementum in hac habitasse platea
2	leo	efcf02ff-f390-4f0f-a5be-060cee2f47ee	aliquet pulvinar sed nisl nunc rhoncus dui vel sem sed sagittis nam congue
3	faucibus cursus	e1de4c7e-585b-48de-9a11-dc12b50f12a2	quam nec dui luctus rutrum nulla tellus in sagittis dui vel nisl duis ac nibh fusce lacus purus
4	nulla facilisi	6f156007-c9b9-42c7-a17f-33cbb917293a	pellentesque viverra pede ac diam cras pellentesque volutpat dui maecenas tristique est et
5	diam neque	1eb29577-6ed4-40c4-8248-5297fbcbaed1	pellentesque quisque porta volutpat erat quisque erat eros viverra eget
6	sit amet	868d001b-e4c0-4cfe-a09d-c11ba8655ee4	in congue etiam justo etiam pretium iaculis justo in hac habitasse platea dictumst etiam
7	nulla dapibus	bc366018-45f0-4431-9aa0-cdc1026b5075	metus vitae ipsum aliquam non mauris morbi non lectus aliquam sit amet diam in magna bibendum imperdiet
8	convallis	f46dfaca-07a8-43c8-8cd6-ebeaaf898172	est donec odio justo sollicitudin ut suscipit a feugiat et eros vestibulum ac est lacinia nisi venenatis tristique fusce
9	lacus	6fcd6b36-6b79-493c-8934-440eb82be7da	pharetra magna ac consequat metus sapien ut nunc vestibulum ante ipsum
10	in	51de3e04-9747-4d4d-8680-68447a920e8d	vestibulum rutrum rutrum neque aenean auctor gravida sem praesent id massa id nisl venenatis lacinia aenean sit amet justo
11	sed vel	1cf46bf0-72e3-4580-a869-b271d0978a46	turpis adipiscing lorem vitae mattis nibh ligula nec sem duis
12	nisl venenatis	3a5b3cf4-3f4e-4d2d-aa7a-55ff2e664bd4	vestibulum eget vulputate ut ultrices vel augue vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere
13	fermentum	ae13bfbf-3341-4d8d-a836-	nullam varius nulla facilisi cras non velit nec nisi vulputate nonummy maecenas tincidunt lacus at

Рис. 4.2. Дані таблиці з ролями

Table CSV

id	name	code
1	lacus at	2a779dac-e5bc-4dd2-a549-2a0d9bbc608f
2	risus	8269871c-90f7-4977-957a-53a6a7990580
3	vestibulum	86522f5f-ecd6-407e-a7aa-51fab481eab6
4	rutrum	0429c3f6-5121-4e2f-96bb-f462ddcff6e7
5	nec	2f9a8fa5-559c-498d-9c80-87ddacca76b6
6	nisl duis	aa82c16c-d155-4f5a-af45-365c3afa7fae
7	venenatis tristique	2f5b40e1-15dc-465b-bedb-37624579850c
8	lorem vitae	984ee3fb-2710-4174-9fe5-436762b1f820
9	amet	bf3a7392-ec26-4aa4-9709-933a0d64080d
10	ipsum	8fd5d5a4-b648-469d-9570-156059e182c0
11	vel	7e33c8b4-1366-4277-b288-729f947b5d74
12	massa	fd8a2d2a-80d0-4cf3-afa4-74a5f4dd739b
13	et	ecae7016-8107-4cc3-a397-1554cae5786d
14	rhoncus	94c359a7-c9f2-4e29-9688-333ed2f7f5db
15	suspendisse	41745e94-3cef-4c90-b79b-7bd8abfd91cc
16	vel	d3d33112-a779-4dc3-96e5-a40d87d8db70
17	tempus vel	bcb50b4d-41ff-47e7-a1f9-608ba8e0d5d3
18	sit	28d6e4c8-47dd-4436-8da7-c51e2bc029e2
19	vel	eed3ceb9-c109-4596-98ae-f4140ed2b6d8
20	tincidunt eu	98e06585-b172-4b5d-95d4-31c02e12f293
21	turpis adipiscing	ab57c7a1-437e-413e-bf09-b81ded3db3f3

Рис. 4.3. Дані таблиці зі статусами

Table CSV

id	name	user_id	description	status_id	structure_id	structure_version
1	varius	1128619c-0ce2-46b5-a8f7-bddd2363e129	pede ac diam cras pellentesque	f20579c3-9022-4ce5-a6bb-29461e13caa0	CAR_MODEL	2
2	quis lectus	594a9952-6c1c-4be9-9e6d-e8ebab44543b	odio elementum eu	c6c22400-bbd0-416a-a17d-8f6cc597c0f8	CAR_MODEL	1
3	ligula sit	843d0c6e-0270-450a-b41b-fae13267f04c	amet lobortis	848bd6d9-7c5c-43d6-a42e-aa183518c18d	CAR_MODEL	1
4	elit proin	86ed6bb8-b23f-41c3-87c7-19f77fe6210d	montes nascetur	90ff3db8-4240-422b-b818-01a0be8e3a9f	CAR_MODEL	1
5	vel	fe0bc461-ca8d-4a2c-a16a-1cce128a1d6a	ut erat curabitur gravida	443b6608-4a6b-4002-9761-1d0dc4400934	CAR_MODEL	2
6	gravida nisi	b5868f8c-7bcc-4602-9bdc-6d1f33c8f869	ante vel ipsum praesent	12ca4ee8-7e91-43c2-9558-3f26d7c5ea5e	CAR_MODEL	2
7	placerat ante	89bb169f-8c5c-46ff-9692-fc1ee898473d	posuere felis sed lacus morbi	fa65dda8-cb4a-4258-9616-fe1e00d8cbc4	CAR_MODEL	1
8	at	dce19104-7f1c-41cc-96b7-bd2ef7e77c6f	quam turpis adipiscing lorem vitae	4da7d0f3-dc15-4a1e-b88e-84f321625eb8	CAR_MODEL	2
9	turpis enim	df7a0c68-eed8-41b0-9be2-e4b6b039edcd	porttitor id	d7a2f39c-1325-42b1-8231-e892fdbe15dd	CAR_MODEL	2
10	sit amet	bd3b3686-e46a-4151-96f6-6479e2a93530	quam pede	4e7b13c7-cb95-4532-a2c6-ec5685355a89	CAR_MODEL	2
11	vulputate ut	6819edcd-01d7-4eb9-94f1-3ab482db3bf9	sed interdum venenatis	f0a5a700-cf8a-4132-a529-b407b3a5ec8e	CAR_MODEL	1
12	vel est	78ca1c76-bf4c-4c00-9687-c0a191672d2a	mauris sit amet eros	ea4bcb49-b76f-40ac-be4c-254a7bdecd77	CAR_MODEL	2
13	bibendum	d782b14b-b0ea-46a3-abaf-	consequat	0e5fdede-29f6-410a-8409-	CAR_MODEL	1

Рис. 4.4. Дані таблиці із оголошеннями

Table CSV

id	participant_id	password	is_locked	is_closed
1	1707fd1e-9faf-471d-9b1d-92635f89ae8a	d6f0012669b7a3e54419836da4f0b999c1d303b1994b0148e7f923efea9e2a25	false	true
2	a2b479be-f93e-4baa-b140-4878d9106624	b9f5eec2bae2ccb72577bb4665b64f1ef0d7021c7e0c4ba6a9e8601a02d14d42	false	false
3	17557107-27d2-4e71-b1c3-d35d0283ab0b	001cbb007297fd7bd7e4b463531589e379f32e8faee04792db63522d04327643	false	false
4	f1c22423-89e3-45e6-b60c-e5f665a23e9d	294f74b8236397fb2f55e8414e619ddf16c36b78df49c8b3cbd409b7733b54b5	false	false
5	0d1bf201-9bcf-46f1-a5cd-3dd5d6b3a4a8	e9dff1fd59dbc42e407d89aa9f35224726591ce278b16b8a225a156007e9d8ea	false	true
6	9ff4ab1c-8a5d-4061-b9c9-462b3a8c6ab5	d5323fcbc0b347cf8fb3e1d7fa3d8fd569ad75a3e8199f24123e873c5842e6d9	false	false
7	2064c858-c222-44ad-a91e-fd61d47248a5	ce29e09081ae6bf8a102c24e1e7ec9eac7482bf9597d44dce9d41895fc13e1b5	true	true
8	e68fb92e-e858-4b30-a279-dcab83821efa	21b5d271378e11ca6a06249710302d40f14f23f8f90cdc1aa68863c09b03bcba	false	false
9	4a3ad761-c2df-4346-93a5-ea35823d7ce3	eb24639ea91a3c07530600a69db15a8ba3501766c5c941d52ac0f3deb14128fa	false	true
10	f658fb1b-a4ff-4958-96ab-cfde5d567388	3b4a40ef7af7e9bd8848b60fb71c6f98a9fc94e92966a03570baed0f47952bc3	false	true
11	a6ca537c-4003-462d-a037-8a241f93dabc	351d2e54f247f8aee906a88b580104640ca57aef905351e658df10bafab607fc	true	true
12	289b8abe-7645-4ab4-9db5-ae1c0958e8b2	4718215efb8a3161cb1a6a6ce1ac0c443a692ba5c9cb731607ea2a352a8fc153	false	false
13	1c6964ae-2060-49a7-ab5c-b8c7863110c3	58df92e7bacd6ecedab64e91b6f9888e306f8611b2aa2e11770ea068a81d64c9	true	false
14	ccf1143a-1f54-43d9-8f6c-208683b8fb4a	f508e8a4c0bc98f4f638e7d77edc82b31f4d2c2352b349f39f00c36b24f9d441	false	true
15	61a6e387-fe9a-4c17-bb50-f59f5cf8d6d6	15e91533a77405043342a797b6c1d5891b6cc0727c809f2bea7be3930f9d685d	true	true
16	f630a1b2-be38-4284-8ccd-bd803ae21508	aa8d9c6bb58d6b16551eb030fd0af5218695ad5d46d65715cbc841ac93f4c1a4	false	false
17	25e5cd7b-cebf-475b-9a14-2cf23fc08770	33bd6f7860cbf040d73d737c0d8b89b1566e22739da1d2f76ddc0a250fcc116a	true	true
18	8cadd756-0b1f-455e-b891-abf29153dddf	c940091421972266345b485c22b3ce9d9582e35900ad5445b9b9e395ee3347cb	false	false
19	05ed14a5-287f-4a45-8c3a-a65de9e41ea5	8d29a36b6a29a27598b2b73e4f72b692f6b78e6674c61649c5054d7216946f7b	false	false
20	3fcc724c-79f5-463d-b557-181f8d18765f	d37e3ed16f5cf005f03e5321996abdeff2b3885d9c81efc73ca7b50606a261b6	false	false
21	2b103a8f-42b9-4e8e-bf7d-94b10e107df2	e17fa62bd7d46d6ccfe1ac808d380d2b37556572210076dc09f941e096761178	false	true

Рис. 4.5. Додаткова інформація для користувачів

Table CSV

id	participant_id	role_id
1	96664694-948f-4dad-b0c6-d5e407bf6cf9	d212d6b3-324f-4de7-94b5-9339dd3a15ac
2	d3425bb9-b869-4ce9-a54c-90eac054cdfb	be407a2e-9a8e-4123-aee9-82a63ebcd464
3	bad2a79b-d929-48de-b0d6-11bda6b62cdd	88b83480-4edf-40e4-b928-752f5ecfd56d
4	592d6bd5-9b8d-4b20-ace9-af13f6c9f438	ec3076be-7651-4a0f-aab8-90e0c6156174
5	e39076fb-d95e-4613-a756-a668ae108eaf	be169c8e-3e00-4d34-8c63-9c494d264d8c
6	6551fa5a-b81a-469b-b34b-460574ead4a2	3d9a655a-87e7-4c1a-8264-54f38c27a3ca
7	a624eec2-f9fb-4136-961b-d6e13c62cab2	953170dd-960e-444e-9c2f-00d8c9106359
8	0e1c63cb-aecb-4625-8599-d83b2a569507	925849c3-121c-4dda-9fba-57dc26669a00
9	bb94e1f9-604f-43bf-b281-6ee9494af9af	60f8430c-9c4c-4e03-a187-9e12c16ca55f
10	d4816dab-c4ac-4869-ae7e-51c395748004	6412310e-9ca0-4964-84a1-ec3e551e64df
11	65a5d148-c040-4d9d-9a2b-dae19b254d45	903562ae-ef61-4c7f-a4f3-1067cba881dc
12	cd94be83-c0a5-443d-96c0-7c0f59787c59	02b14e03-27d7-4189-a3bb-544bb49a452a
13	d5abfb54-6c12-4432-b8c5-f914290965cb	71af5ae0-8517-438c-a920-e6a83514f198
14	7cccbc04-09e0-4202-8c24-bf3cf2e21ee2	b81f616b-f456-4c4c-808e-26e3ba183456
15	2c6ec693-b21f-421a-b37c-c88fa046585b	4c73d5c9-ea37-4a08-9cf9-f54082b78da7
16	d4a43d8d-7659-4bf7-b5a1-68d55b99a0a1	af6848ba-ef41-4f91-869c-55dd32e78e15
17	6a8b149f-b8ed-4477-abd3-c3ac311e3343	abaca7ac-7ec4-486d-b6f1-fce7f1b713ba
18	d58e334d-97a6-4606-bfd3-fbb72902d811	3ed8381b-0c0f-460d-b83d-fd10a110bcb7
19	f4b5c0e9-11c2-4f6a-ba54-ba16b7b81825	fa9faa2c-f598-4648-883e-7a3756926532
20	0fd9cdbc-b2d9-4e63-9075-ad96fcb4c5df	d2938fc3-1540-421a-ba8c-272563a52f28
21	94ff08e0-3e99-4c33-9cc3-7f9708390075	66296792-d3e0-4d37-9227-ef4f13f7cda9

Рис. 4.6. Зв'язуюча таблиця користувача із його ролями

4.2. Тестування API кожного із мікросервісів

4.2.1. Налаштування системи клієнтом

Спершу буде виконана конфігурація системи. Першим кроком користувачу потрібно завантажити валідні ролі, які він хоче бачити у системі.

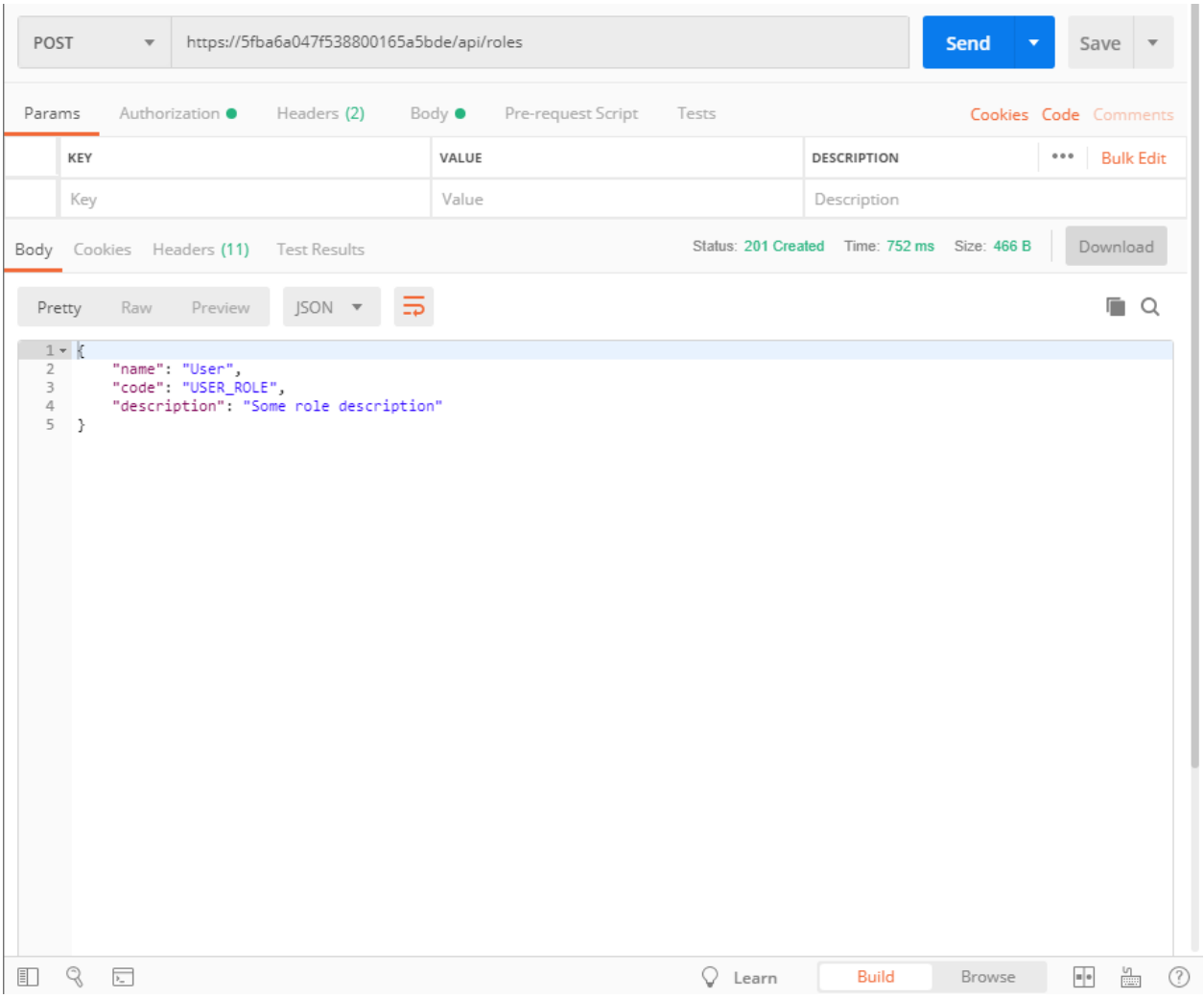

Рис. 4.7. Успішне додавання нової ролі

Після завантаження усіх необхідних ролей, платформ інженер може отримати список усіх актуальних ролей у системі.

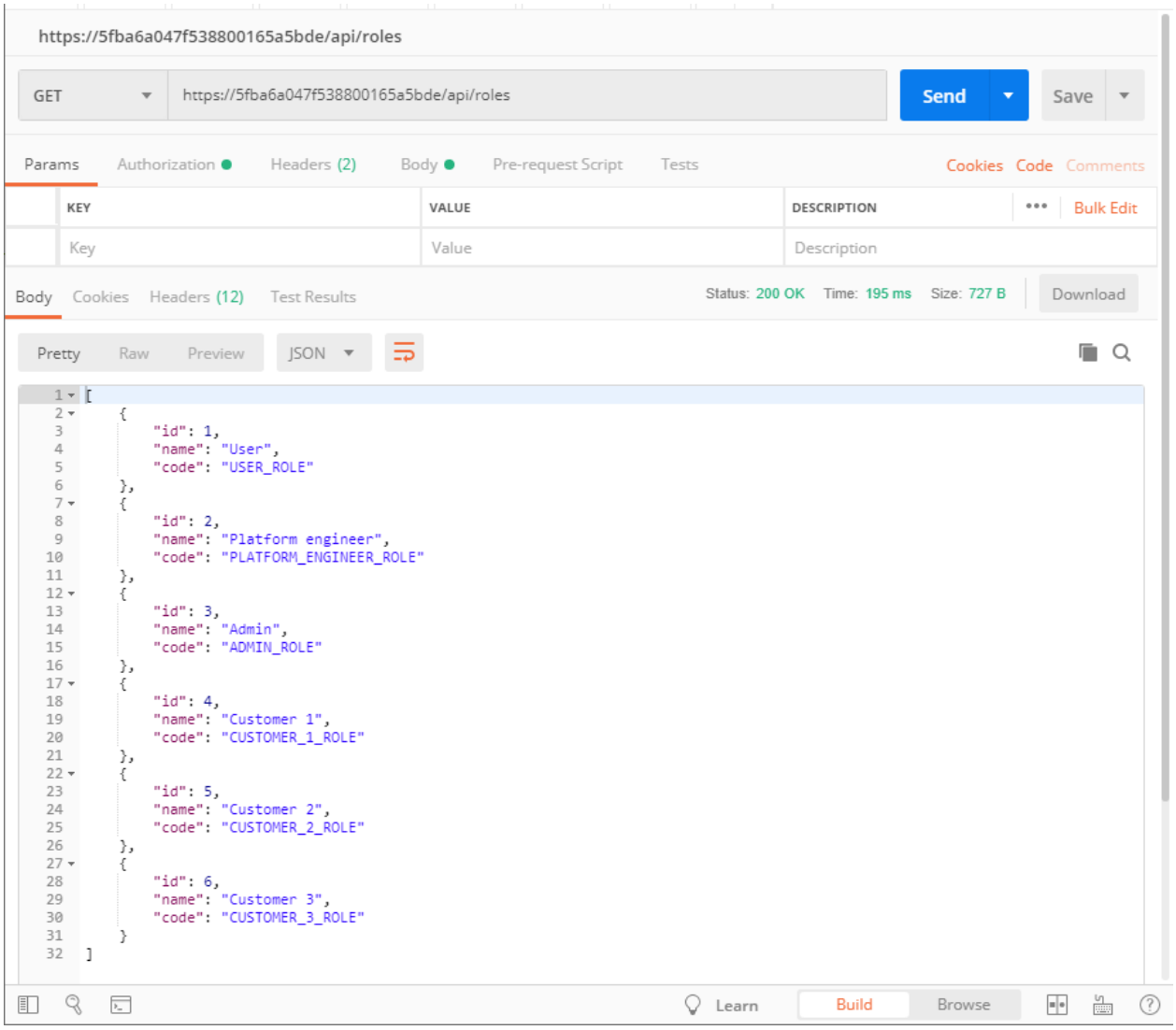

Рис. 4.8. Список усіх завантажених ролей у системі

Тепер замовнику потрібно виконати налаштування структур даних автівок. Потрібно задати структуру даних полей оголошення, також можна виставити необхідні валідатори, задля забезпечення безпечного клієнтського вводу даних.

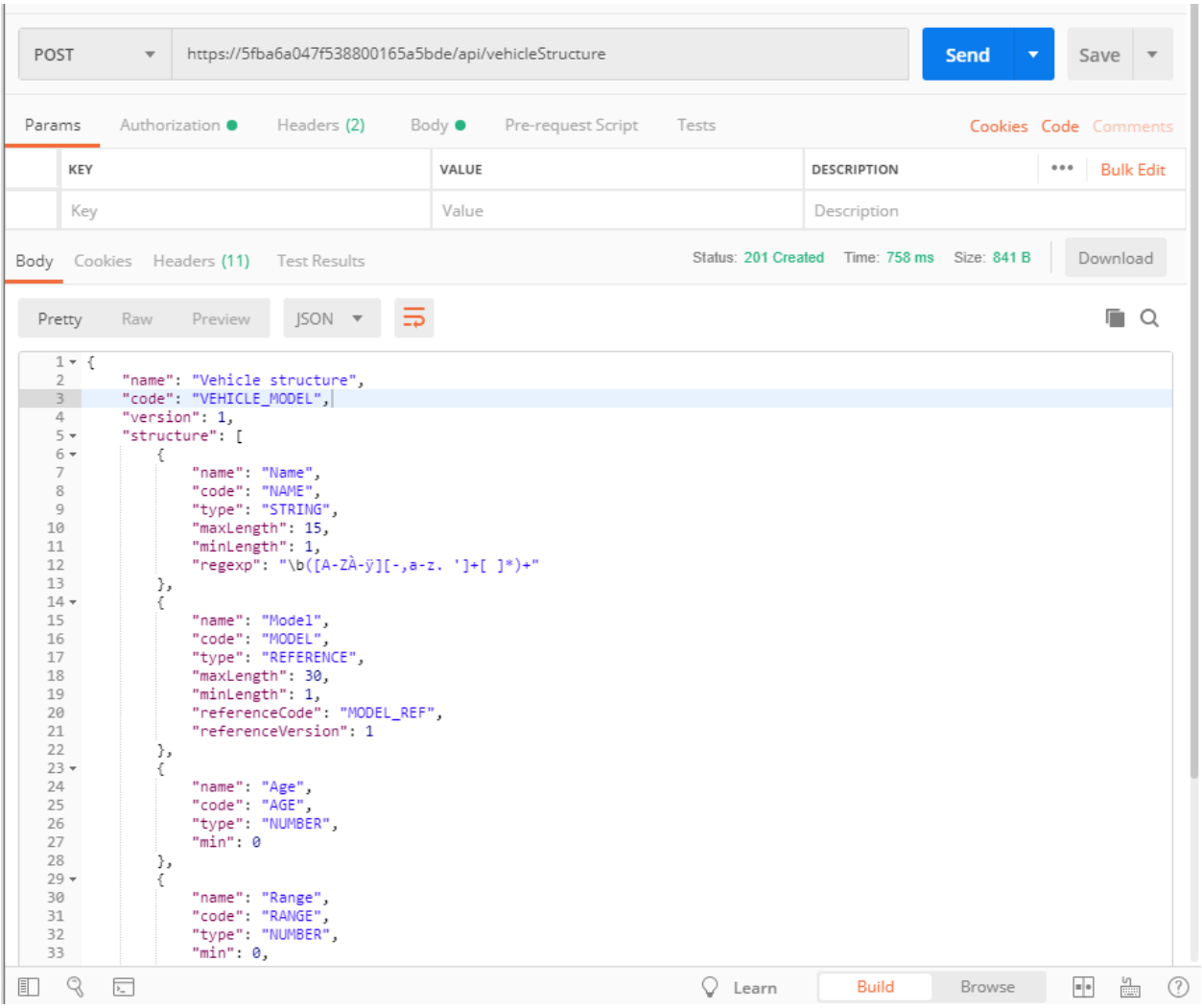

Рис. 4.9. Створення структури даних автомобілю

Після успішного створення структури ми можемо отримати список використаних структур.

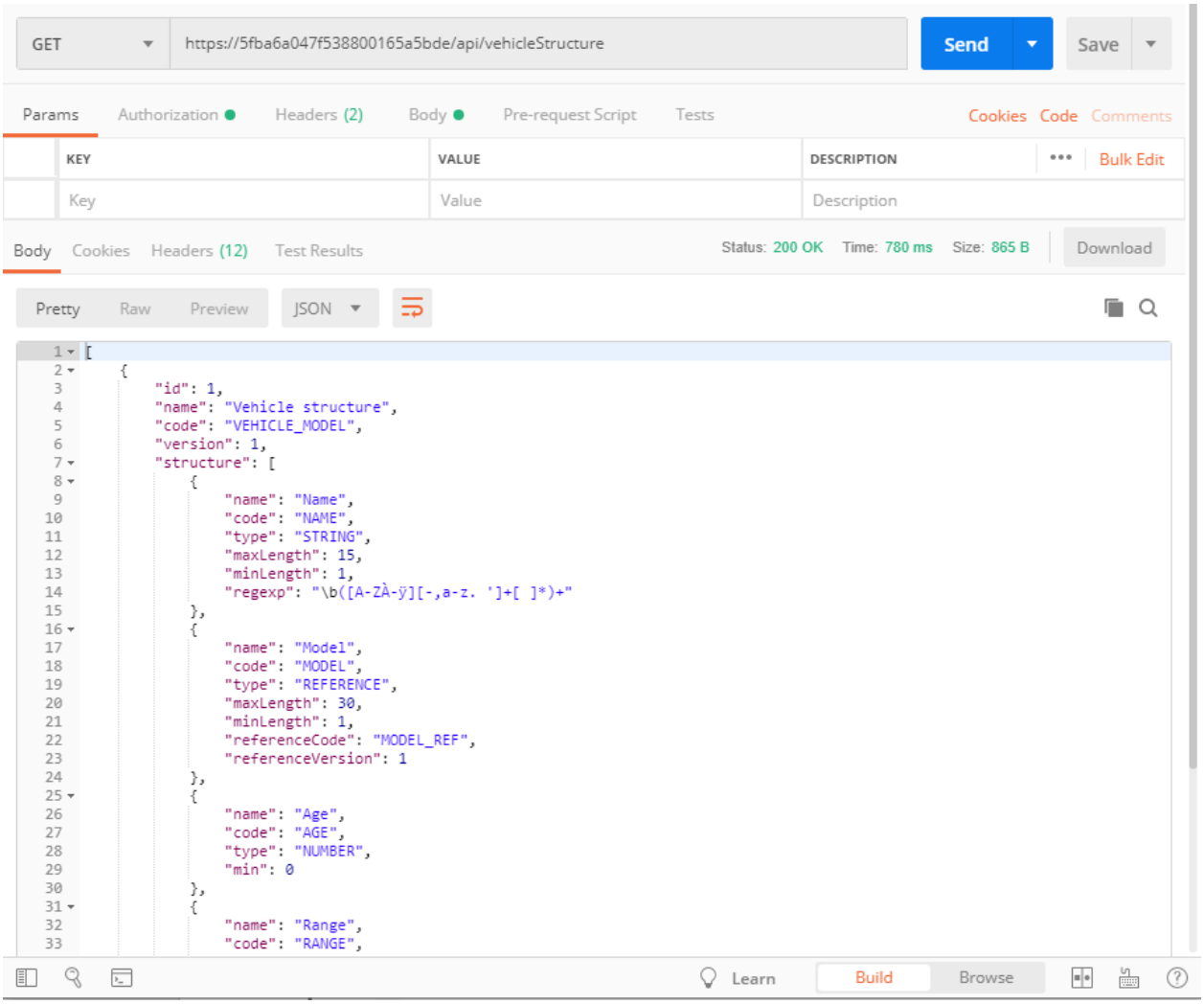

Рис. 4.10. Список успішно завантажених структур даних оголошень

Останнім обов'язковим кроком для конфігурації рішення є завантаження усіх доступних стейтчартів оголошень.

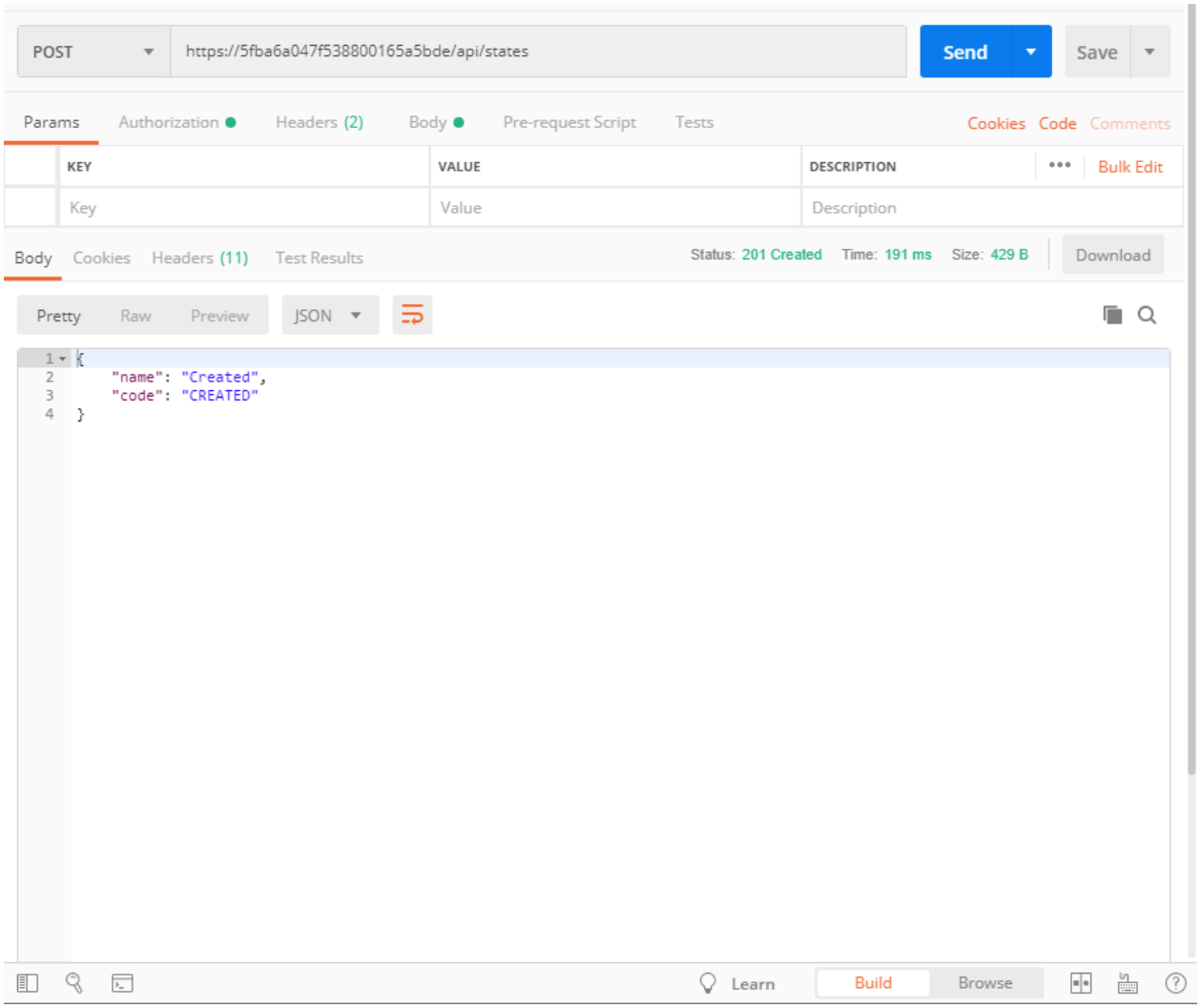

Рис. 4.11. Створення нового статусу

Отримання усіх доступних статусів.

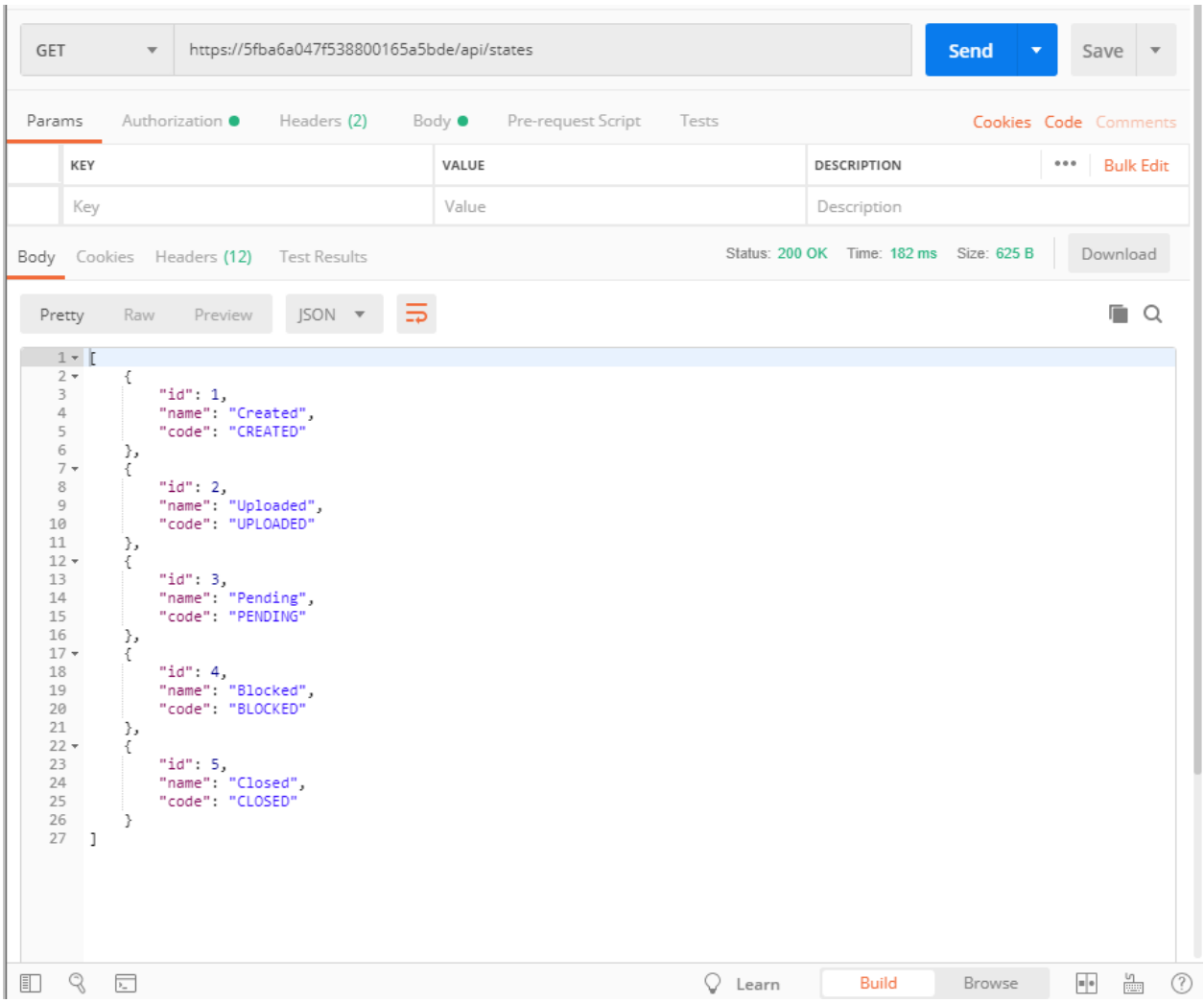

Рис. 4.12. Доступні статуси у системі

Тепер замовнику потрібно задати механізми переходів статусів. Фактично кожен із статусів має деревовидну структуру та може переходити в інші. Завдання задати цю конфігурацію використовуючи наданий API.

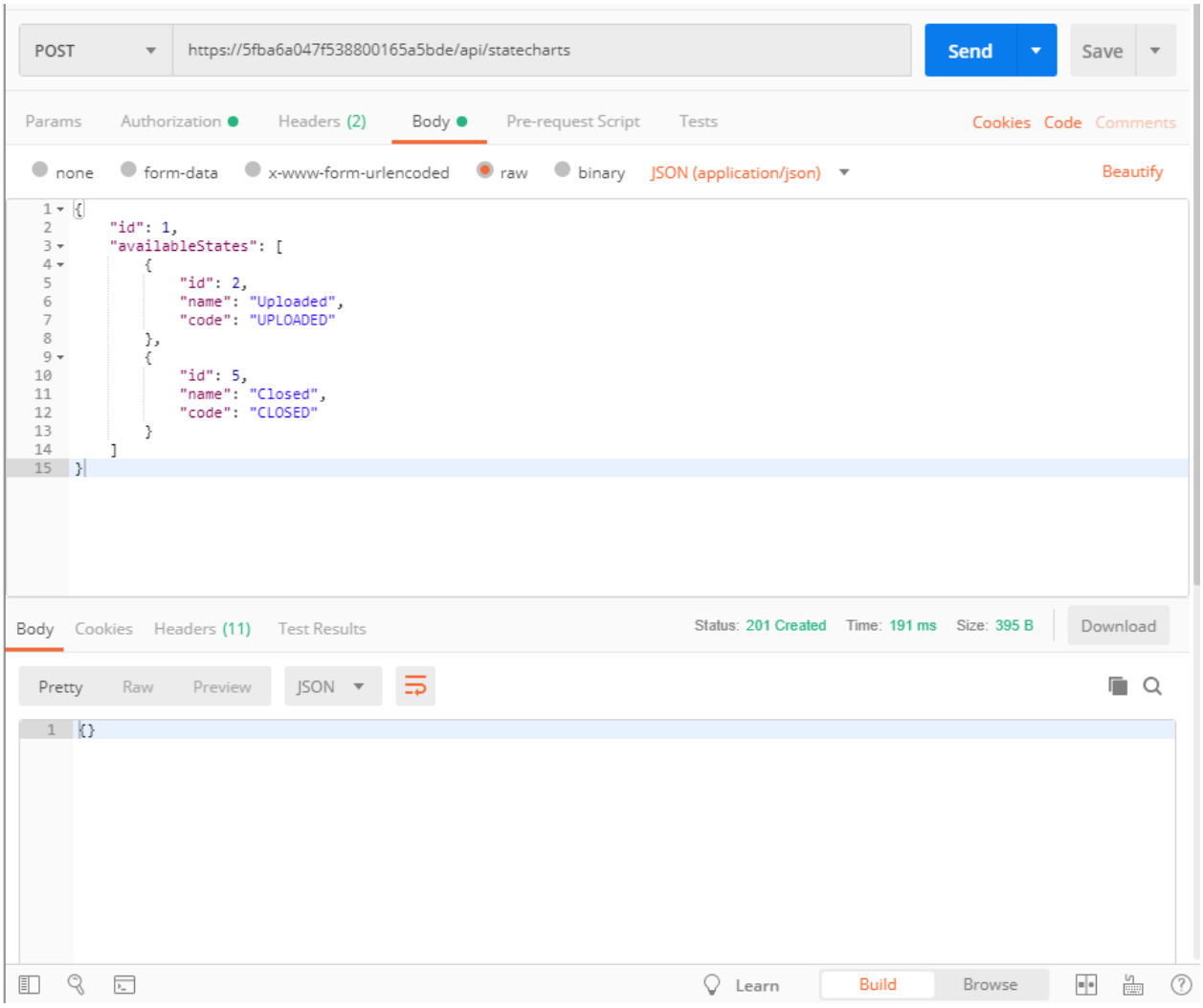

Рис. 4.13. Результат додавання нового переходу між статусами

Замовник може отримати по кожному із статусів усі доступні переходи.

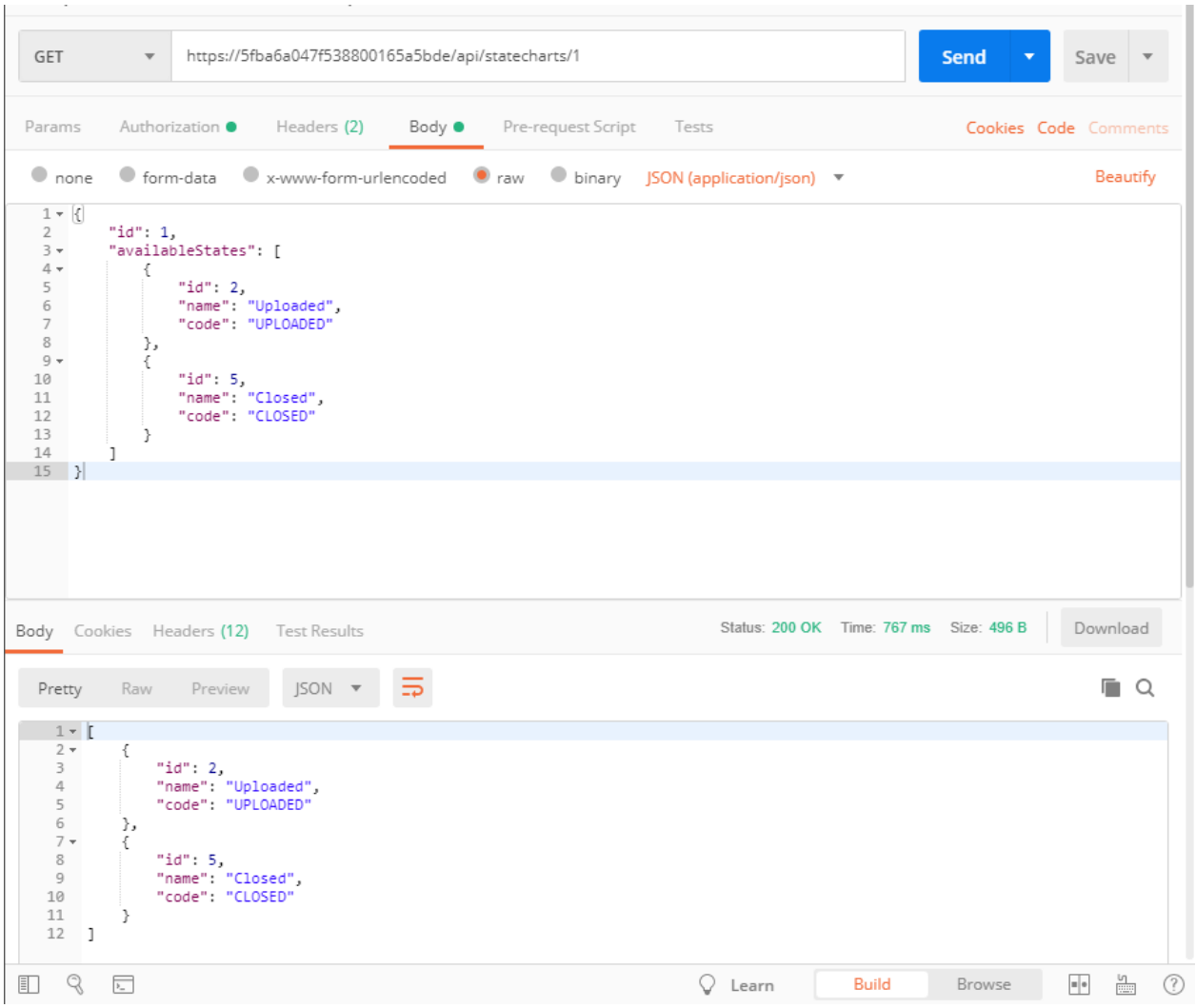

Рис. 4.14. Доступні переходи між статусами для статусу з ідентифікатором 1

По факту обов'язкові кроки для розгортання платформи уже виконані.

Проте є ще декілька доступних опціональних варіантів для більш гнучкої кастомізації. Це налаштування агрегацій пошуку/фільтрів та структур профілей користувачів.

Спершу розглянемо кастомізацію механізмів пошуку.

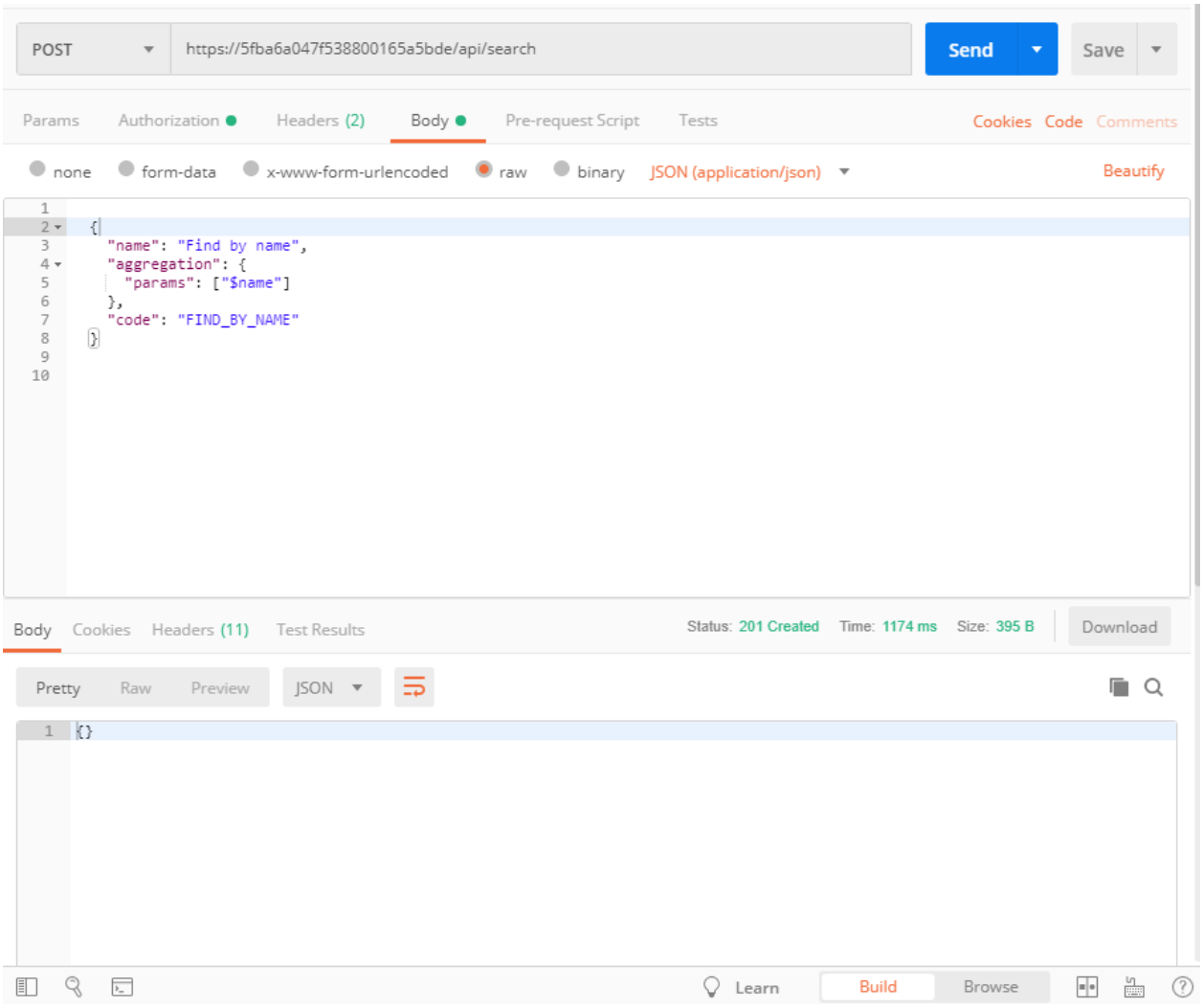

Рис. 4.15. Створення нового фільтру для пошуку

Після створення усіх необхідних фільтрів, ми можемо отримати загальну кількість.

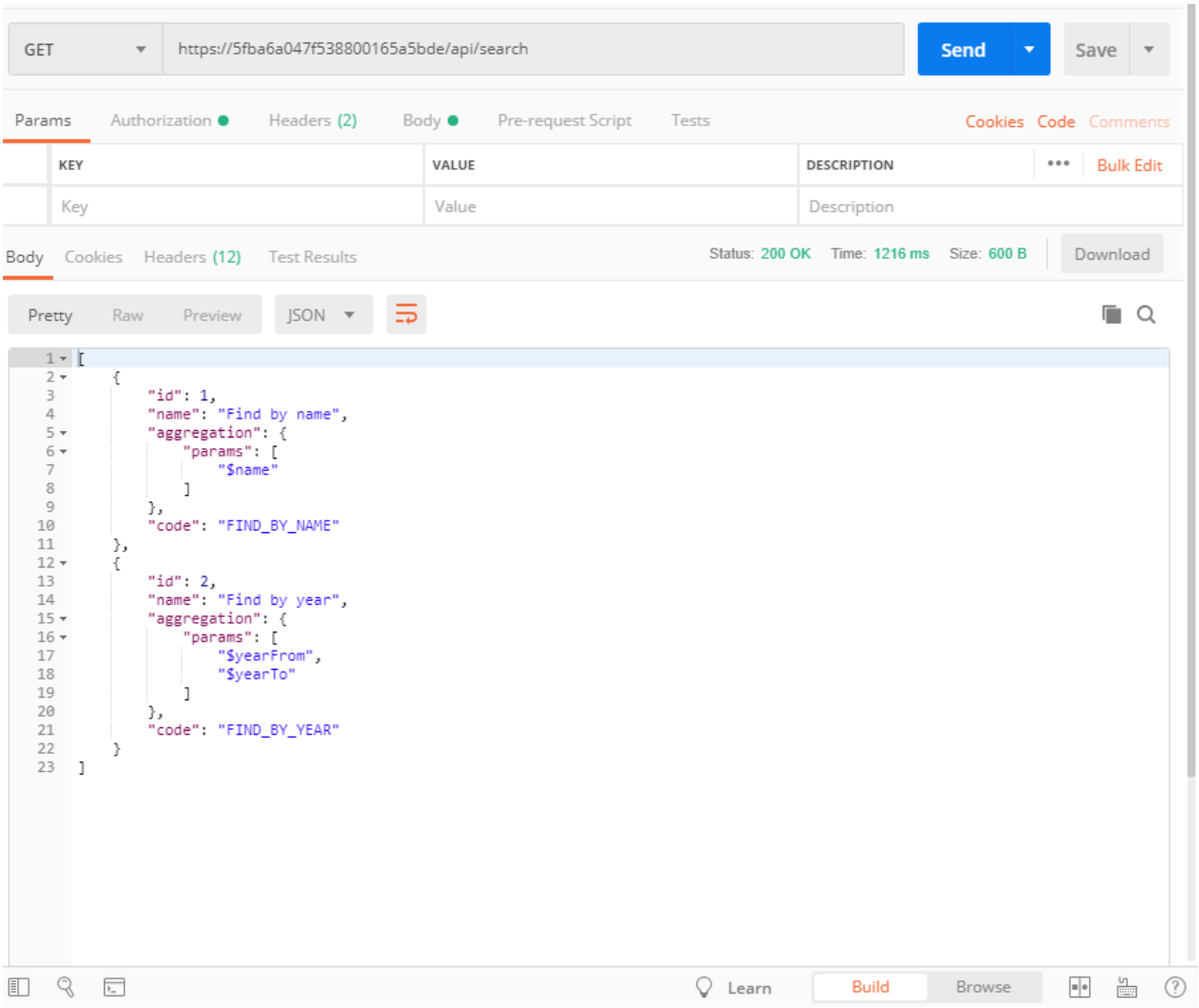

Рис. 4.16. Список доступних фільтрів

Можна оновити існуючий фільтр.

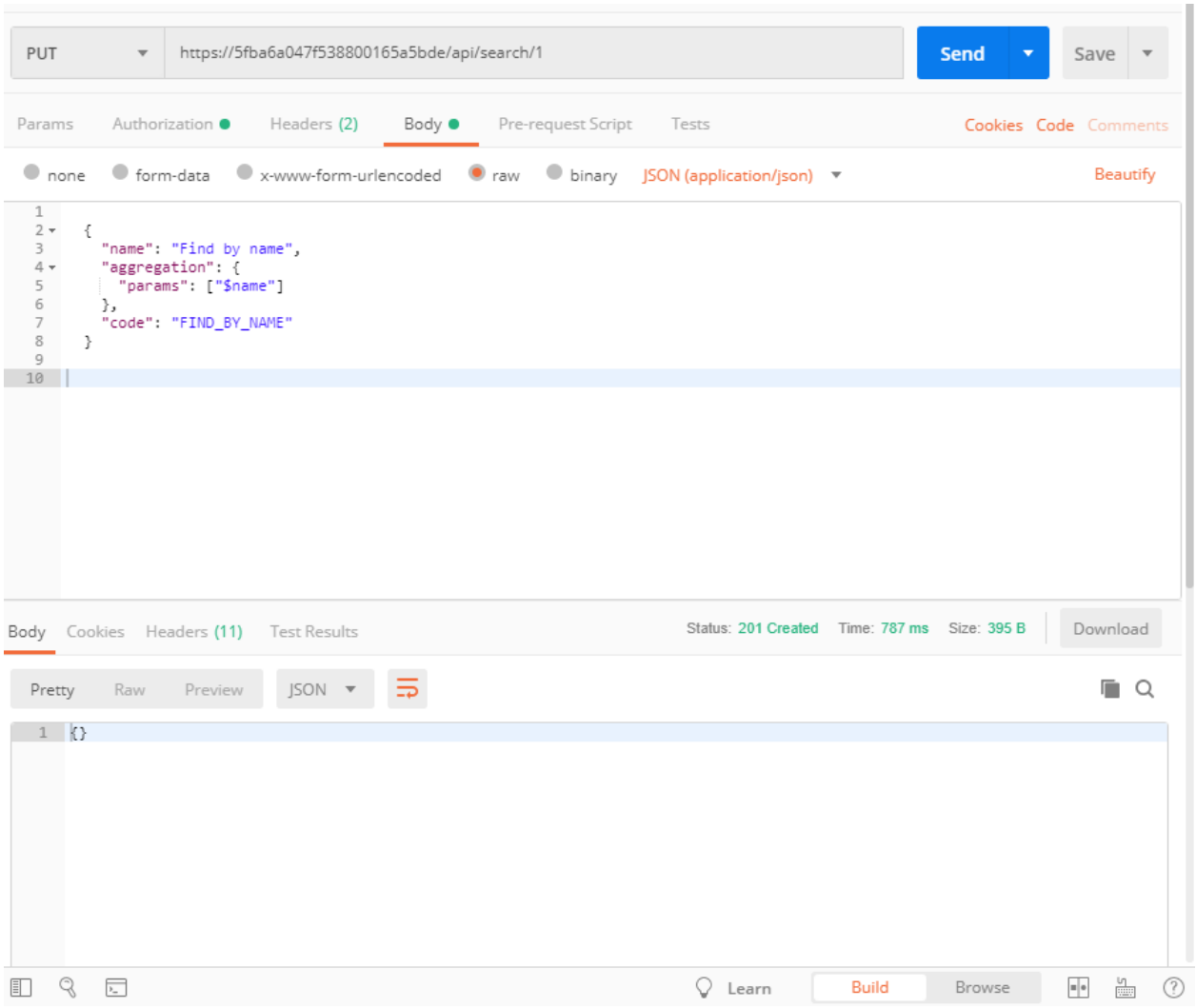

Рис. 4.17. Оновлення існуючого фільтру

Тепер налаштуємо структуру профілю для користувача. Надання структури профілю є схожим до задання структури оголошення, ми описуємо список доступних полів. Оскільки цей крок опціональний, якщо структура не буде задана то буде використана структура за замовчуванням.

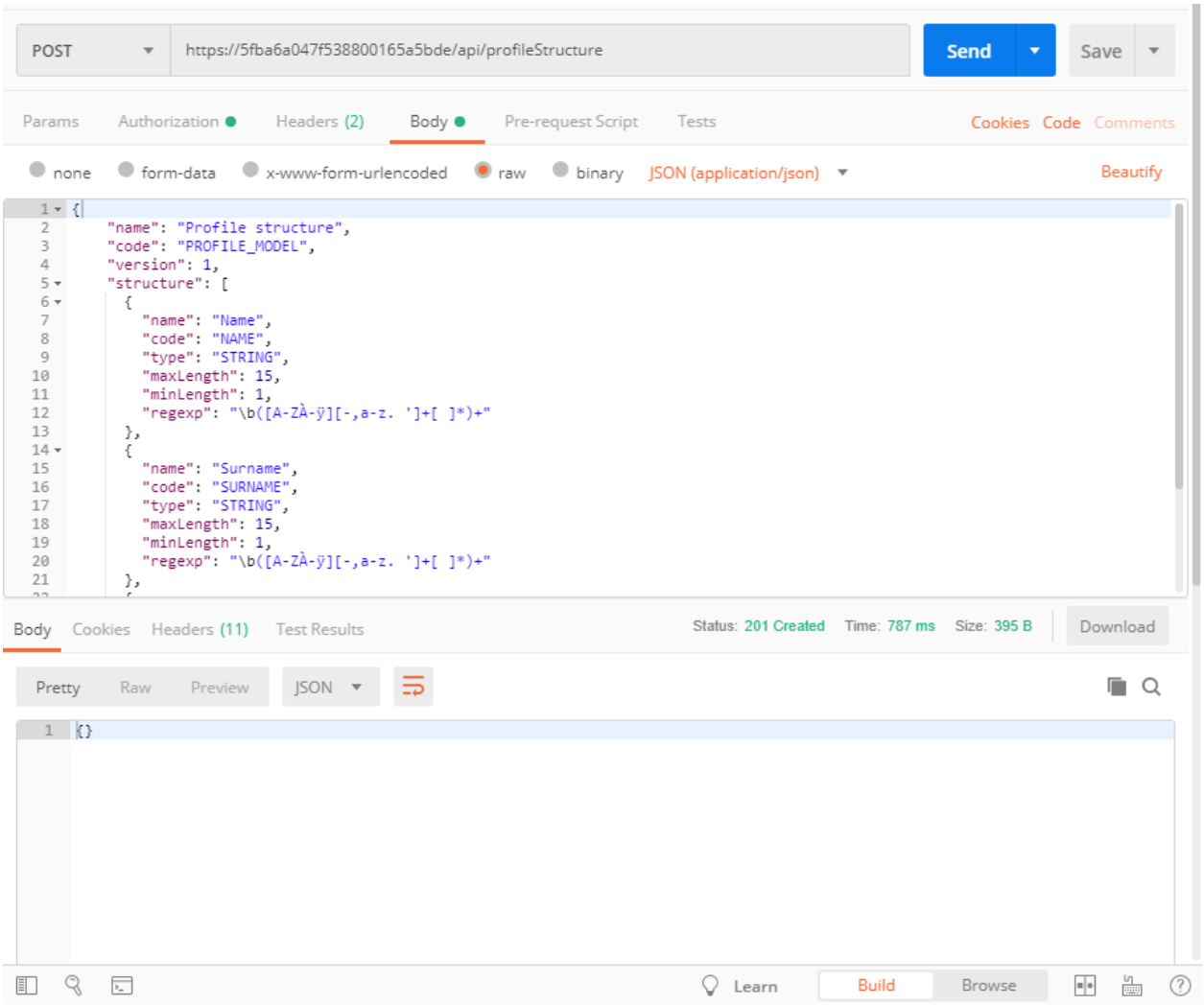

Рис. 4.18. Створення структури профілю користувача

Замовник може оновити та видалити завантажену структуру.

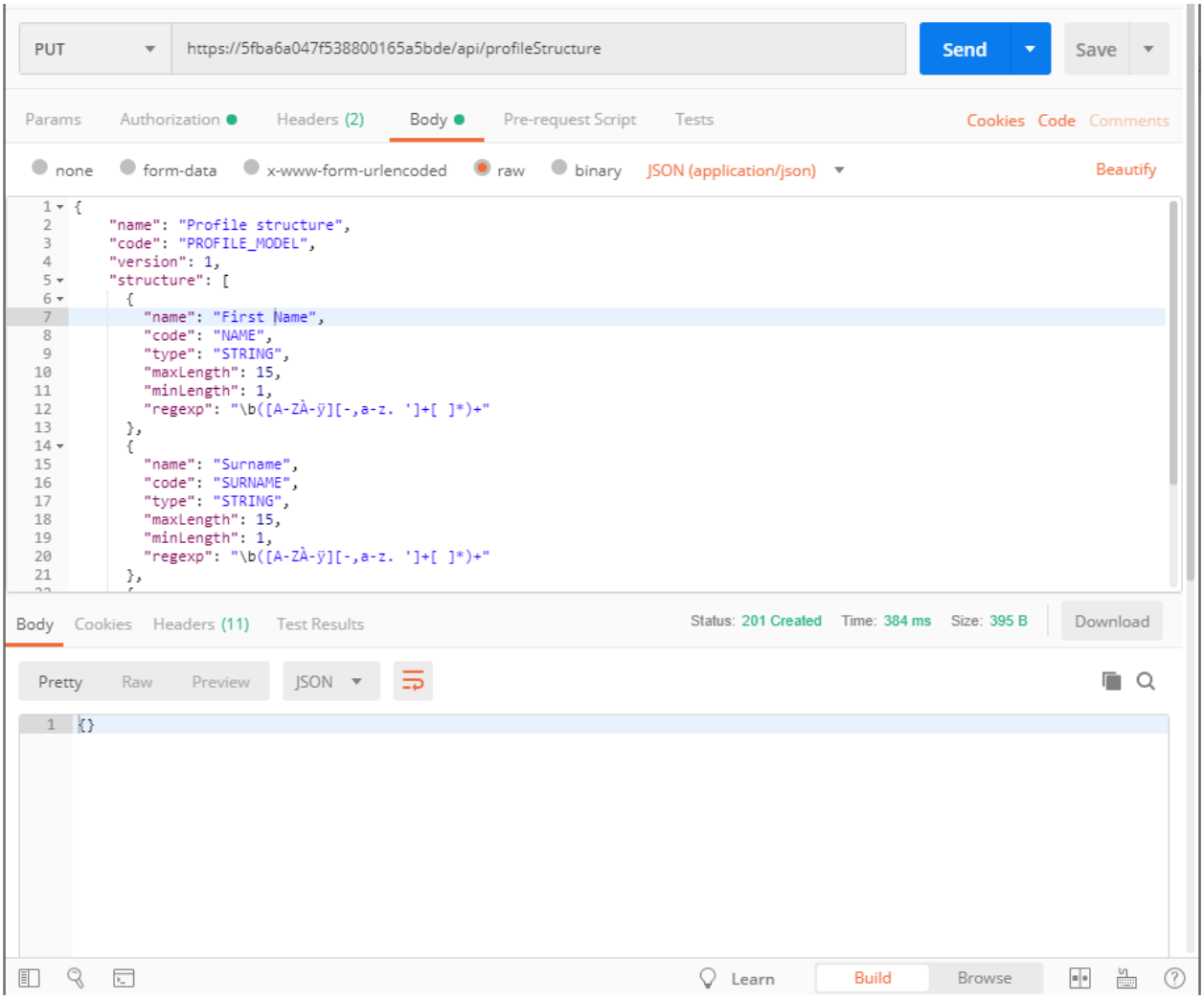

Рис. 4.19. Запит на оновлення структури профілю користувача

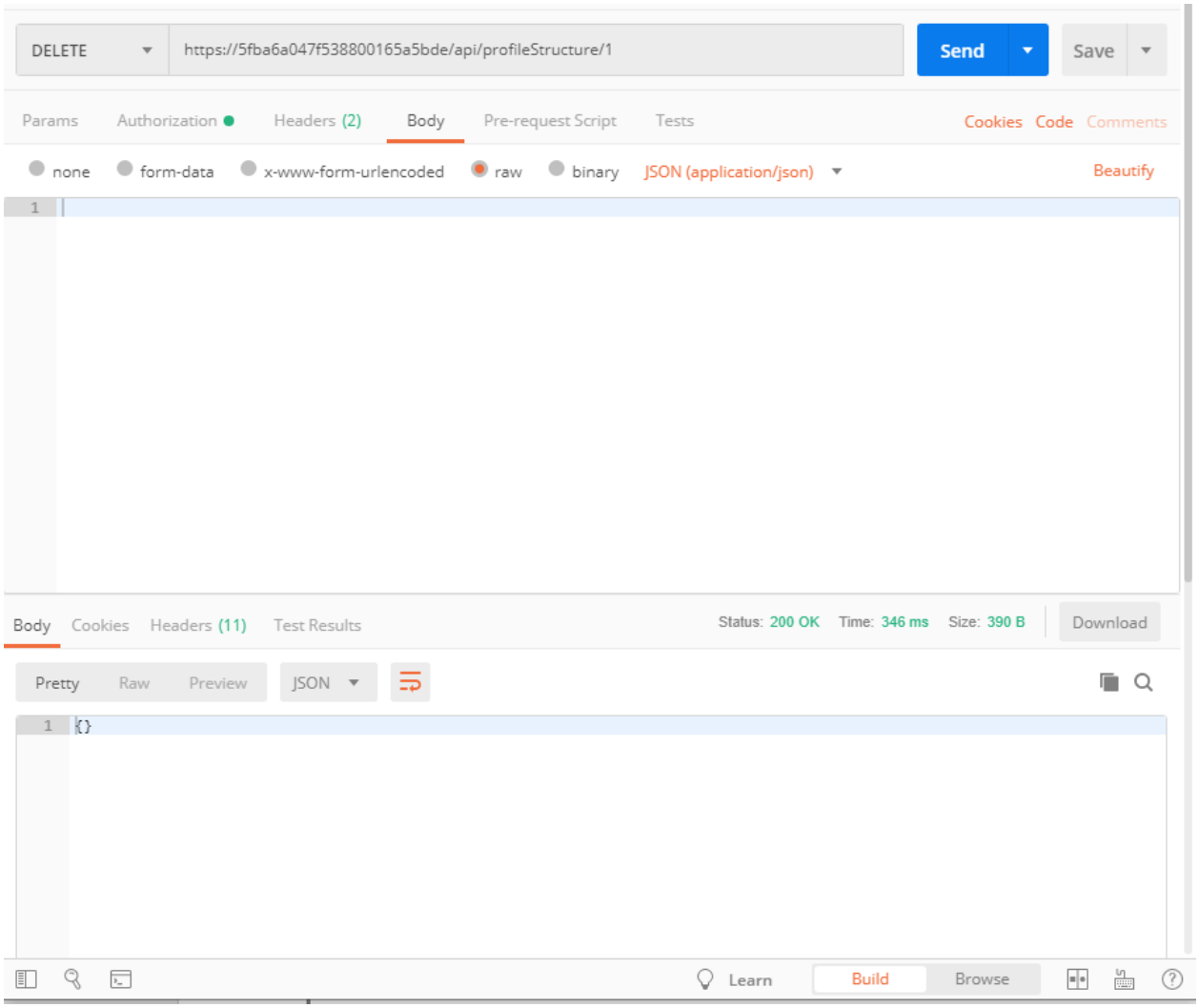

Рис. 4.20. Видалення структури профілю користувача

Також користувач має можливість переглянути список усіх доступних стурктур профілей користувачів.

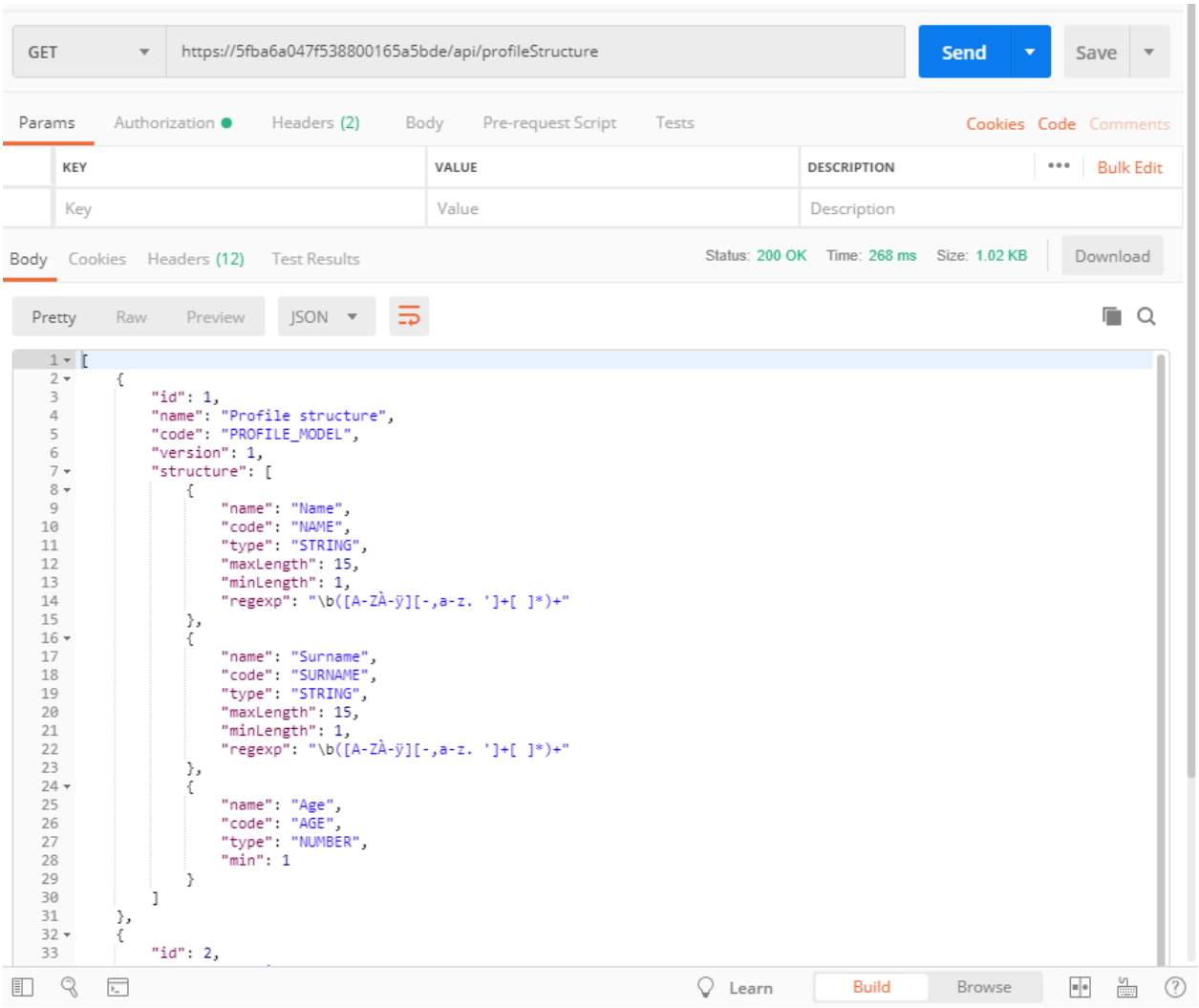

Рис. 4.21. Список доступних структур профілів користувачів

Користувач може отримати профіль по ідентифікатору.

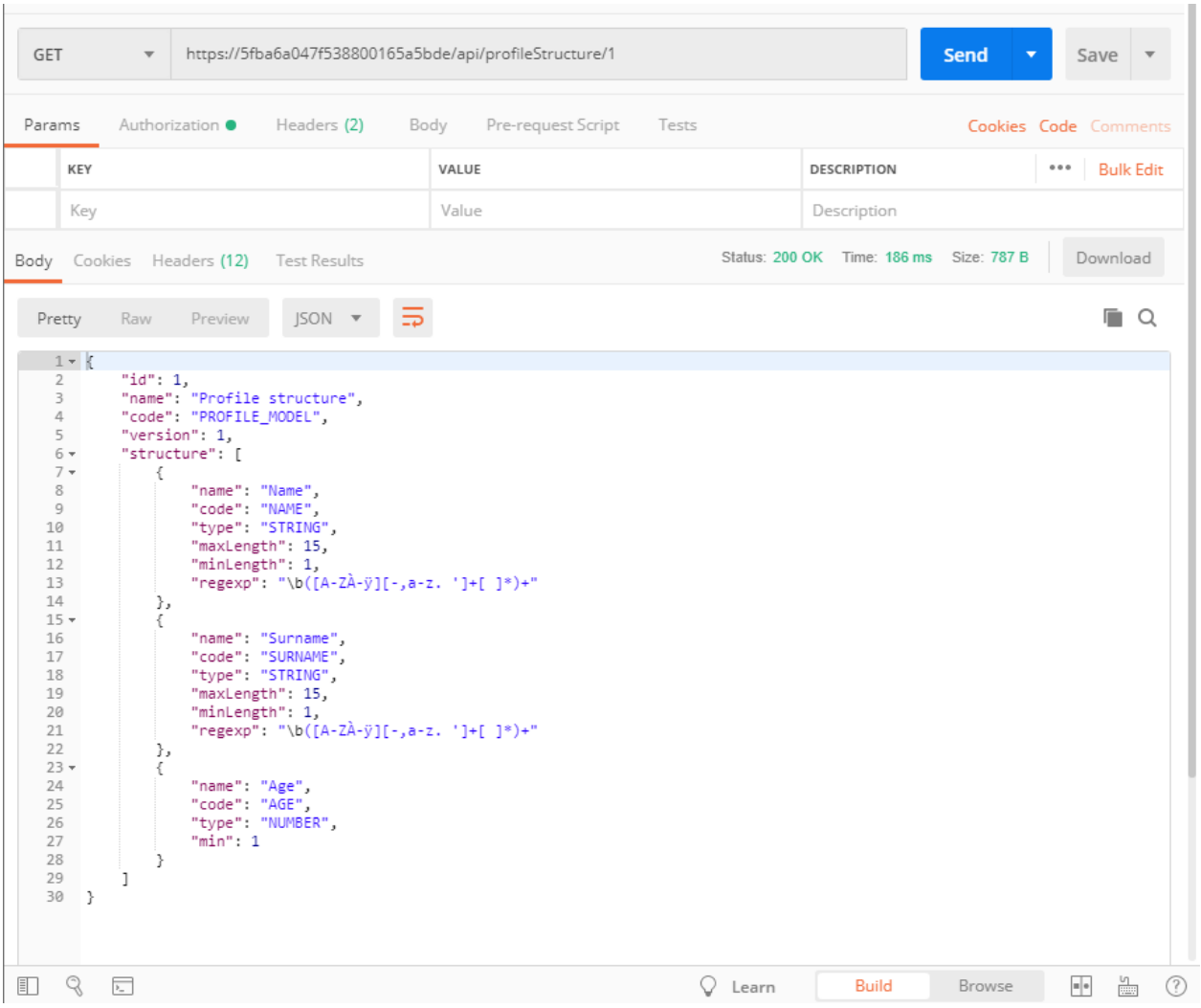

Рис. 4.22. Структура профілю по ідентифікатору

Ще одним опціональним кроком є налаштування нотифікацій та їх темплейтів.

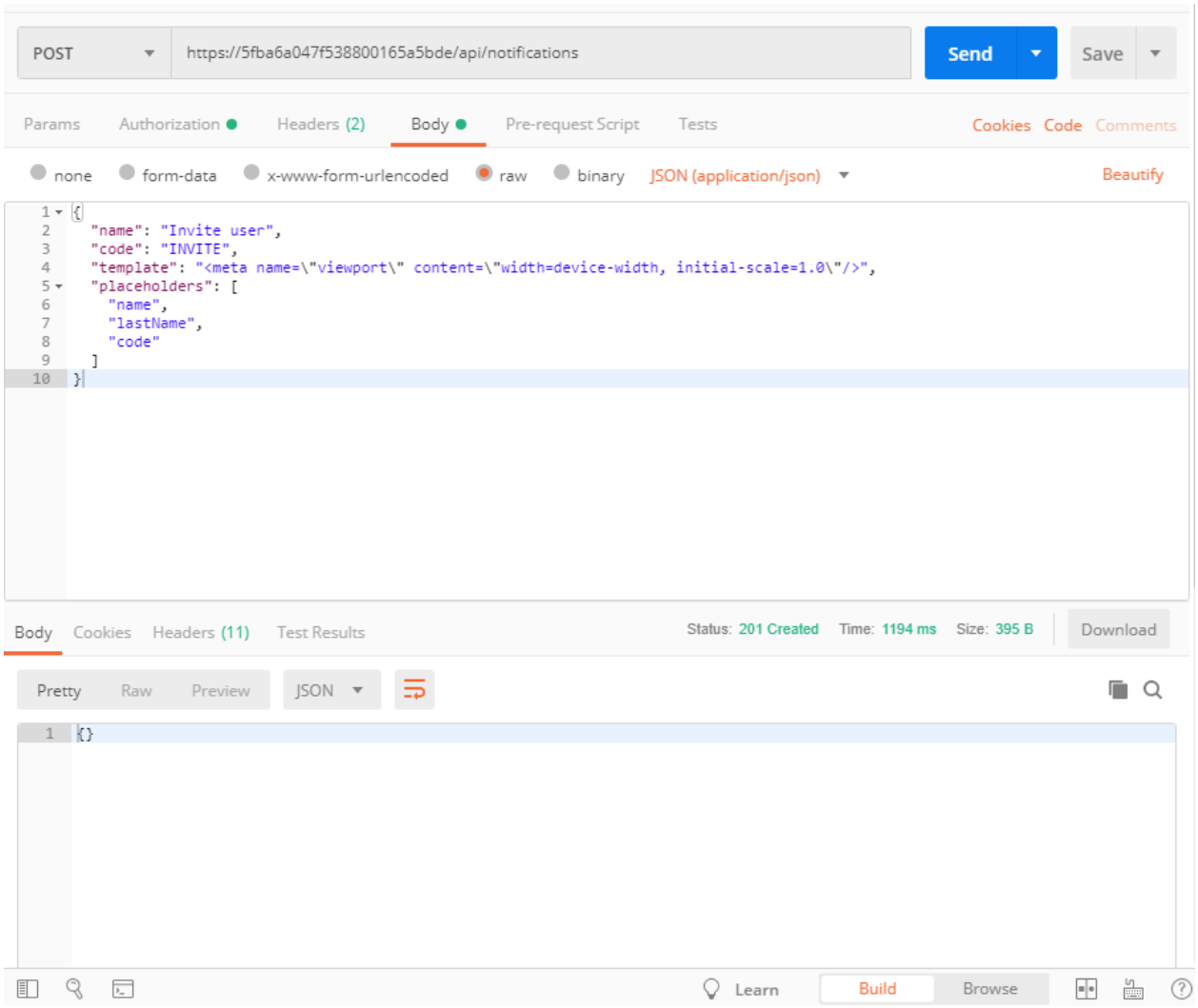

Рис. 4.23. Створення нового темплейту нотифікації

Як і любий інший ресурс ми можемо оновити та видалити темплейт. А також отримати темплейт по його ідентифікатору.

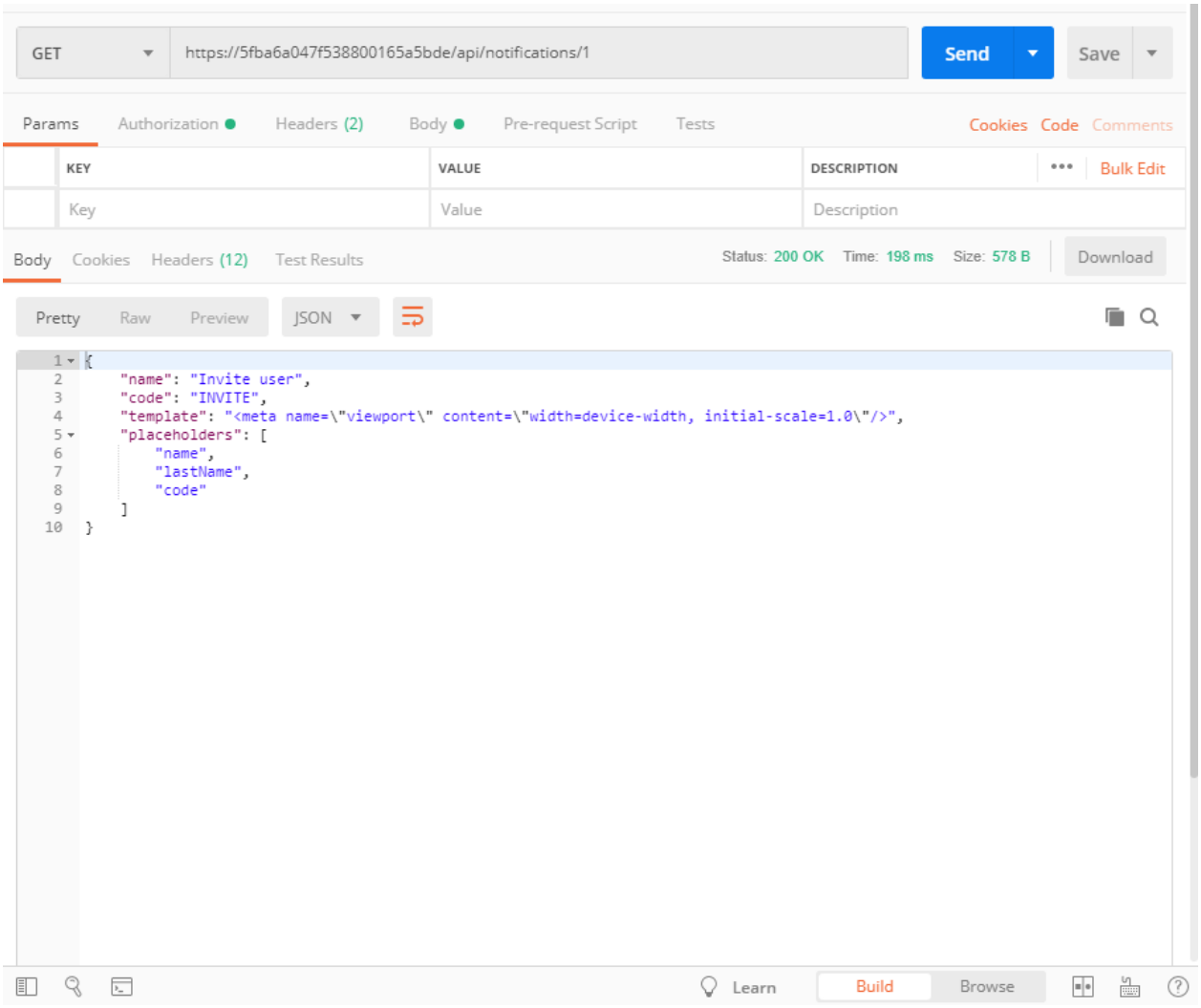

Рис. 4.24. Вибірка темплейту по ідентифікатору

4.2.2. Тестування системи користувачем

Після розгортання уже налаштованої платформи, користувачі мають змогу повноцінно користуватись усіма ресурсами які були кастомізовані конкретним замовником.

Першим кроком буде створення нового користувача.

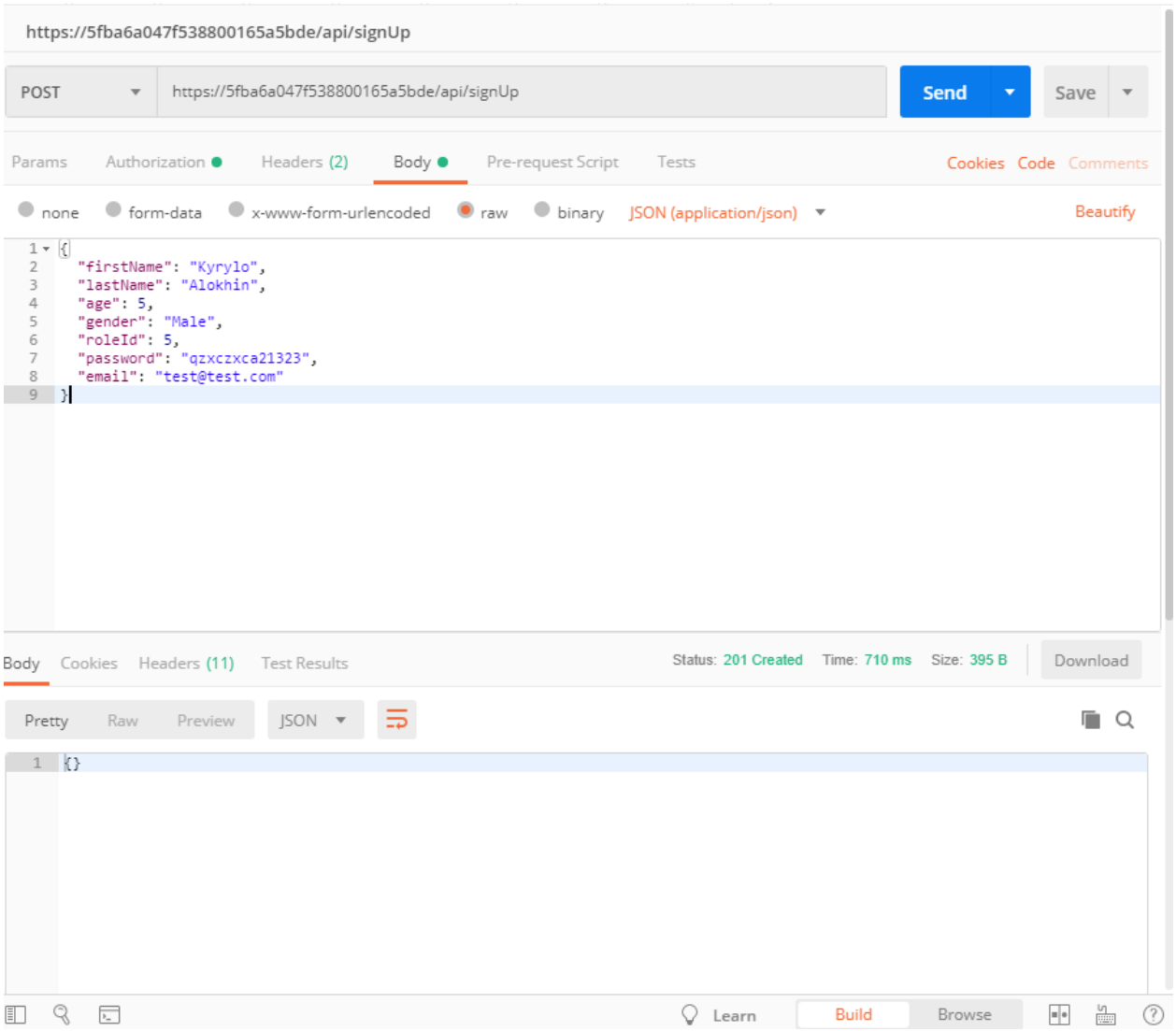

Рис. 4.25. Реєстрація користувача

Для підтвердження реєстрації користувач повинен перейти по посиланню яке він оримає на свою електронну пошту.

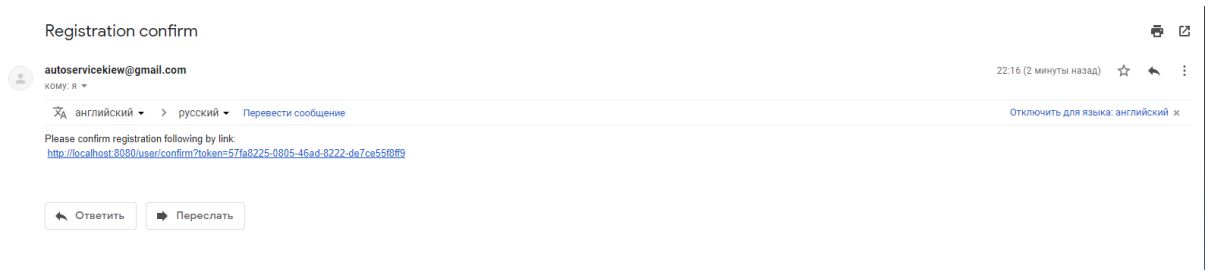

Рис. 4.26. Підтвердження реєстрації користувачем

Зареєстрований користувач має можливість створювати/оновлювати/видаляти свої оголошення.

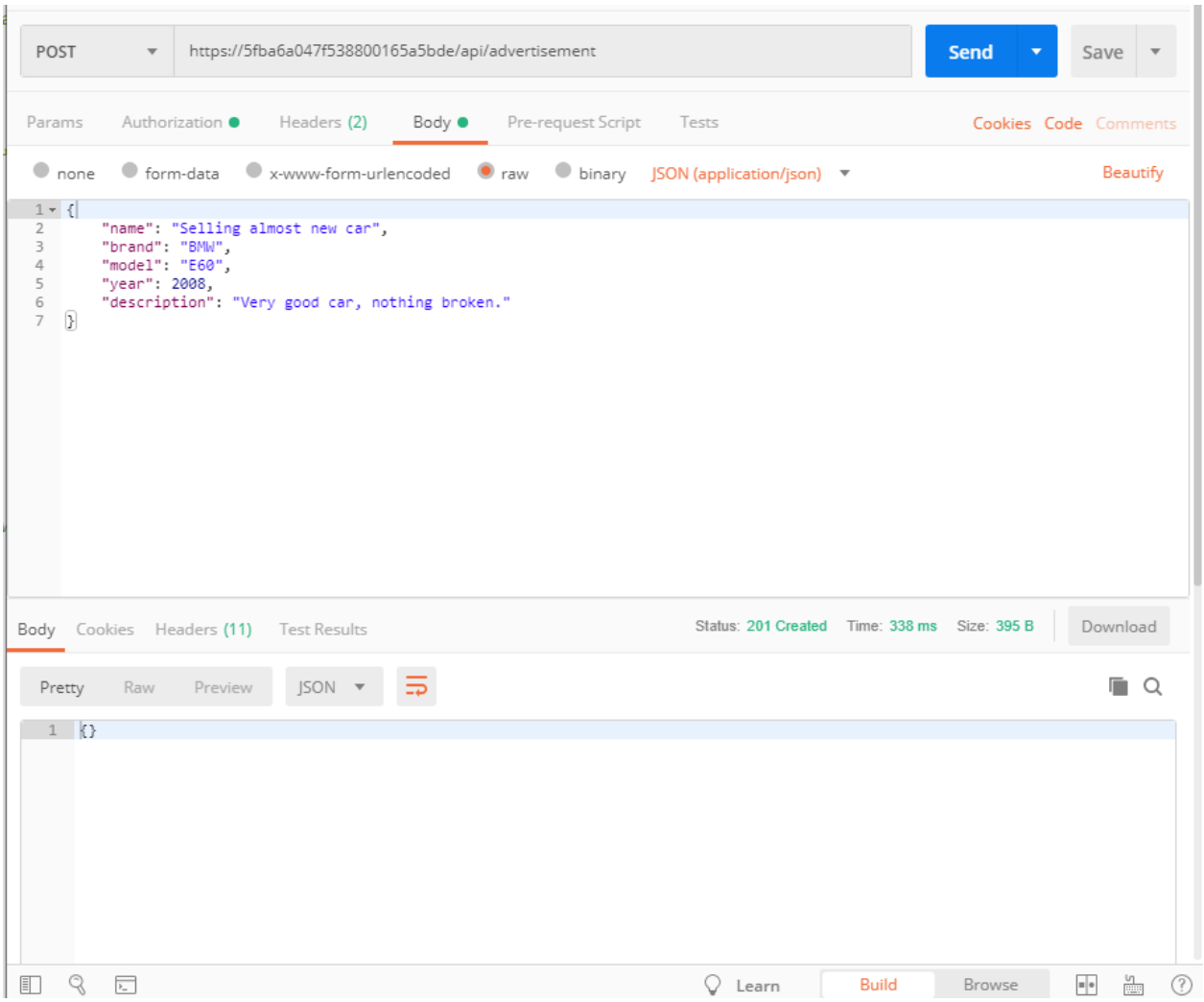

Рис. 4.27. Створення оголошення

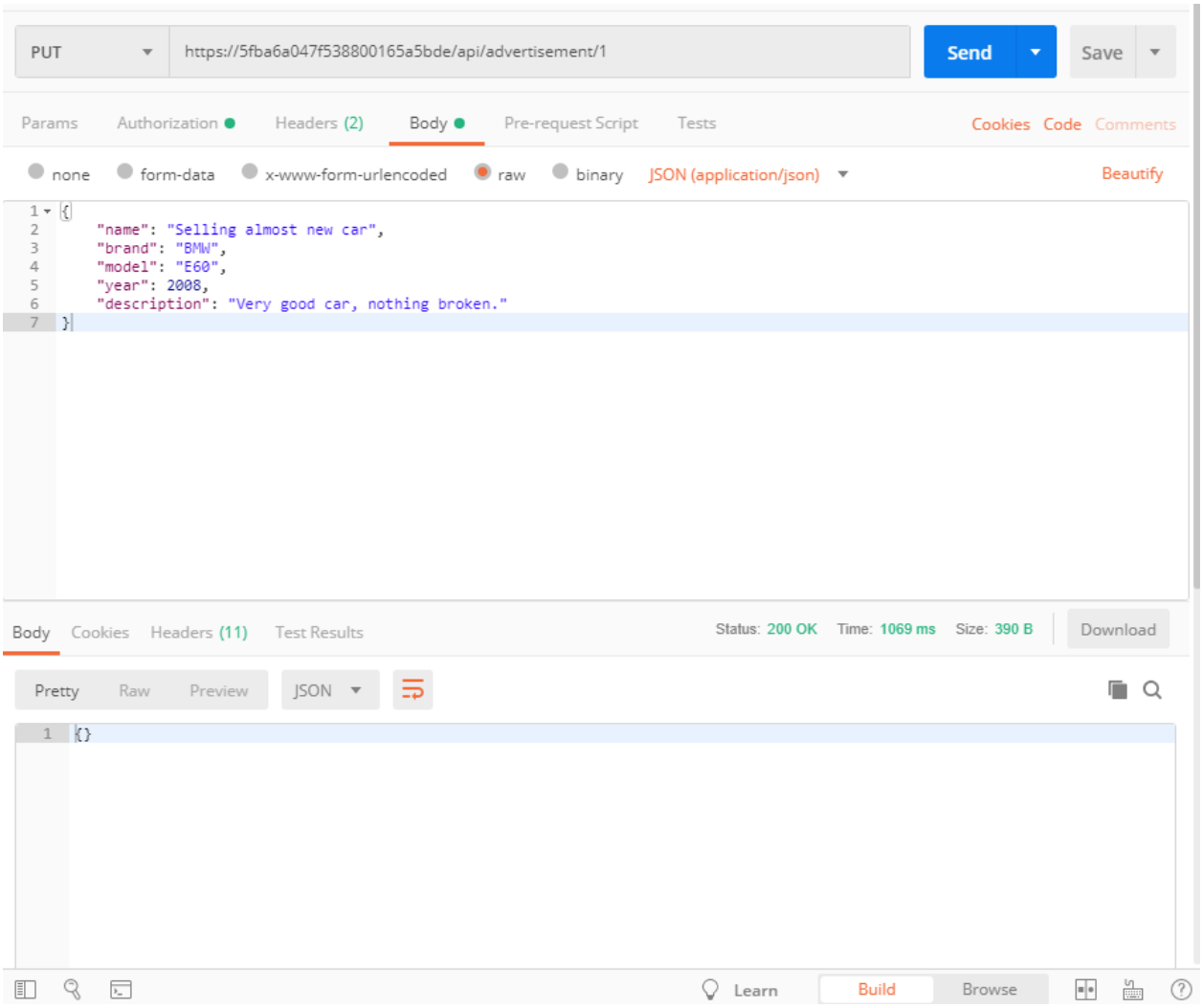

Рис. 4.28. Оновлення оголошення

Отримання списку усіх доступних оголошень.

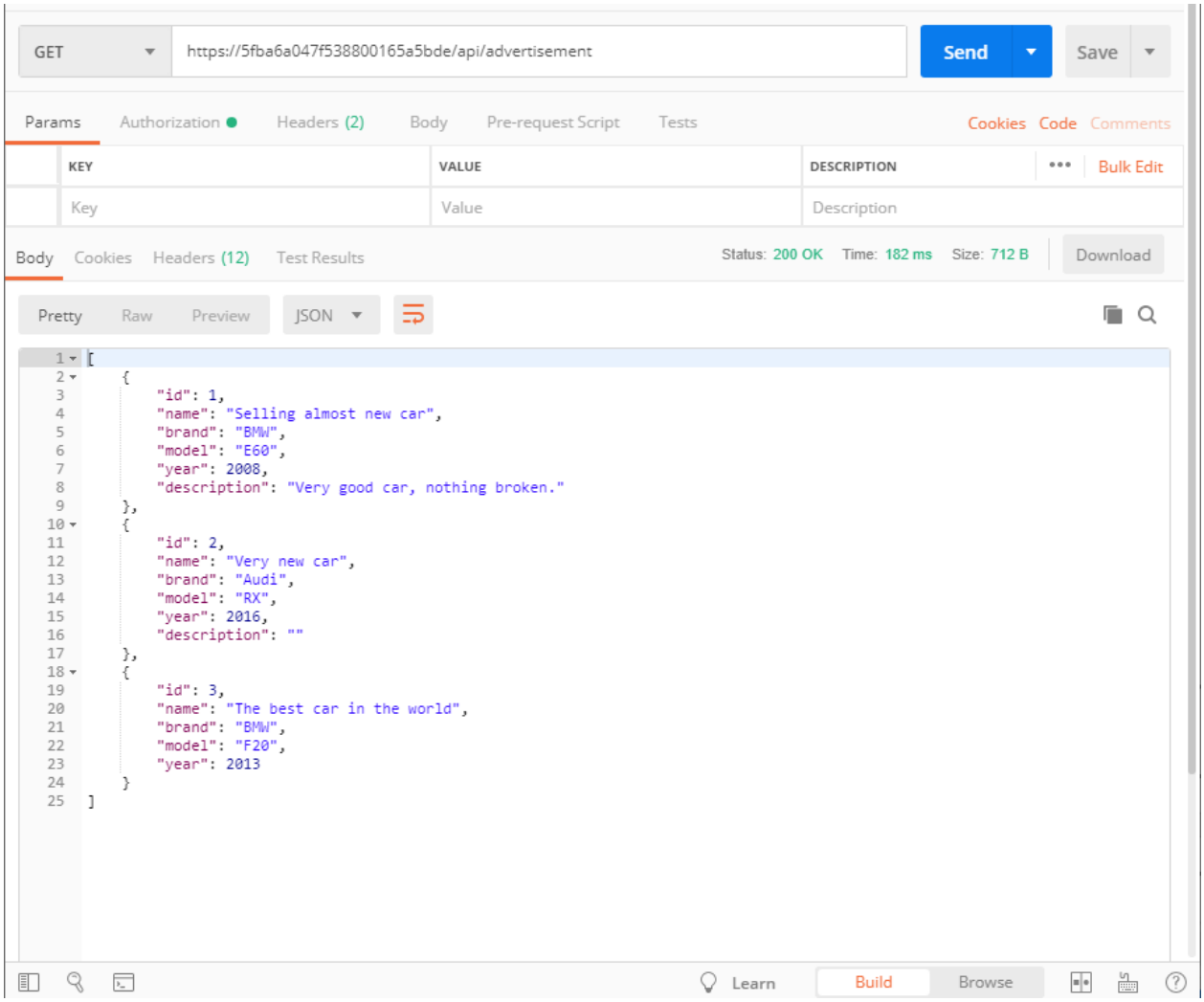

Рис. 4.29. Список усіх доступних оголошень поточного користувача

Висновок до четвертого розділу

На основі реалізації платформи ми можемо виділити основні частини конфігурації:

- Налаштування ролей
- Налаштування стурктур моделей транспортних засобів
- Налаштування агрегацій для фільтрації
- Налаштування статусів та їх переходів

Також для розгорнутої платформи користувачу буде доступний наступний функціонал:

- Реєстрація
- Вхід в систему
- Пошук та фільтрація оголошень згідно доступної конфігурації
- Створення/оновлення/видалення оголошень
- Взаємодія з оголошенням (переходи по статусам)

РОЗДІЛ 5
РОЗРОБЛЕННЯ СТАРТАП ПРОЕКТУ

5.1. Опис ідеї проекту

Проект являє собою багатомодульну платформу для трейдингу транспортними засобами. Основна ідея полягає у тому, що клієнт може динамічно налаштувати продукт під себе, та використувувати його у своїх бізнес цілях.

Основні якості продукту:

- Надає можливість конфігурації одного рішення специфічного під кожного клієнта;
- Конфігурація є динамічною, а отже може змінюватись під час роботи додатку;
- З використанням мікросервісної архітектури, ми легко та гнучко можем доповнювати функціонал платформи новими компонентами та частинами;

Наприклад, є декілька клієнтів, котрі зацікавлені розширити свій бізнес по продажу автівок на ринок інтернету. Клієнт формує заявку на підключення до платформи. Клієнт має змогу ознайомитись із основним функціоналом платформи, у випадку якщо він хоче підключити додатковий функціонал, він комунікує з бізнес та маркетинг командою котра оцінює наскільки даний функціонал належить платформі, чи для конкретного клієнту. Далі іде юридична частина, підписання необхідних контрактів згідно послуг. Наші інженери реалізовують функціонал, (якщо потрібно) котрий потрібен конкретному клієнту. Далі, згідно вимог, виконується налаштування рішення під конкретного клієнта та наступне розгортання платформи на хмарній інфраструктурі (Amazon Web Services, Azure, Google Cloud Platform).

Нижче, у таблиці 5.1, наведений опис ідеї стартап проекту.

Таблиця 5.1

Опис ідеї стартап проекту

Зміст ідеї	*Напрямки застосування*	*Вигоди для користувача*
Багатомодульна платформа трейдингу транспортними засобами	1. Торгівля транспортними засобами 2. Стартапи у сфері обміну та трейдингу товарами 3. Реклама нових товарів та їх подальша утилізація 4. Адаптивне рішення під кожного клієнта	1. Ринок клієнту розширяється за рахунок інтернет ринку 2. Динамічна конфігурація в режимі реального часу 3. Підтримка рішень специфічних під кожного клієнта

Згідно з вище наведеної таблиці, видно, що даний стартап має досить широкий набір для застосування.

Отже, навіть невеликі клієнти мають змогу розширити свій внутрішній ринок, за рахунок мого рішення.

За рахунок динамічної конфігурації, клієнту не потрібно адаптовуватись під реалії чужих рішень, він має змогу зберегти усі специфіки ведення його бізнесу. Це також дозволяє скоротити час інтеграції із платформою та перехід на інтернет торгівлю.

Нижче наведено аналіз сильних та слабких сторін для мого рішення. Для цього описана таблиця 5.2.

Таблиця 5.2

Визначення сильних, слабких та нейтральних характеристик ідеї проекту

№ п/п	Техніко-економічні характеристики ідеї	(потенційні) товари/концепції конкурентів			W (слабка сторона)	N (нейтральна сторона)	S (сильна сторона)
		Моя платформа	Autobahn-tech	Wizzle			
1.	Вартість використання програми	Середня	Висока	Середня		+	
2.	Функціонал додатку	Значний	Значний	Незначний			+
3.	Динамічна конфігурація	Так	Так	Ні			+
4.	Локалізація	Так	Ні	Так		+	
5.	Підтримка кастомних клієнтських рішень	Так	Ні	Ні			+
6.	Мобільний додаток	Так	Ні	Так			+
7.	Веб додаток	Ні	Так	Так	+		

Отже, можемо зазначити, що ідея додатку несе досить велику кількість переваг, поміж поточними рішеннями на ринку. Основним недоліком можна винести відсутність веб додатку, що може бути запланованою активністю для реалізації на майбутнє.

5.2. Технологічна звітність проекту

Потрібно проаналізувати технологічний стек, для реалізації ідеї. Нижче наведена таблиця 5.3 описує необхідний стек.

Таблиця 5.3

Технологічна реалізація проекту

№ п/п	Ідея проекту	Технології її реалізації	Наявність конкурентів	Доступність технологій
1	Додаток, що надає клієнт-специфічне рішення для організації процесу продажу транспортних засобів	Було принято рішення використати мікросервісну архітектуру, мову програмування Java, та основний стек для ентерпрайз розробки: Spring Cloud.	Схожі рішення наявні для Американського ринку за винятком певних опцій, описаних вище.	Доступна

5.3. Аналіз ринкових можливостей запуску стартап проекту

Розглянемо основні ринкові можливості для реалізації стратап додатку, що можна використати для впровадження та винесення продукту на ринок. Наведемо структуру попередньої характеристики аналізу понетційного ринку стартапу у таблиці 5.4.

Таблиця 5.4

Попередня характеристика потенційного ринку стартап-проекту

№ п/п	Показники стану ринку (найменування)	Характеристика
1	Кількість основних конкурентів	4 (Autobahn-tech, Wizzle, Carwago, Swap Motors)
2	Загальний обсяг продажів, грн/	-
3	Динаміка ринку (якісна оцінка)	Зростає
4	Наявність обмежень для входу (вказати характер обмежень)	Відсутні
5	Специфічні вимоги до стандартизації та сертифікації	Відсутні
6	Середня норма рентабельності в галузі (або по ринку), %	>60%

В таблиці 5.5 визначені групи потенційних клієнтів, та список потреб, що формують ринок.

Таблиця 5.5

Характеристика потенційних клієнтів стартап-проекту

№ п/п	Потреба, що формує ринок	Цільова аудиторія (цільові сегменти ринку)	Відмінності у поведінці різних потенційних груп клієнтів	Вимоги споживачів до товару
1	Потреба в розширенні ринку продажу товарів, підняття дохідності та кількості активних клієнтів	Малий та середній бізнес	Різним клієнтам доступний загальний функціонал додатку а також можливість розробки специфічного функціоналу.	Чітке формулювання необхідних вимог. Консультація при налаштуванні або самостійне налаштування на основі документації.

На основі аналізу характеристик потенційних клієнтів, можемо провести аналіз основних загроз (Таблиця 5.6) а також можливостей (Таблиця 5.7).

Таблиця 5.6

Фактори загроз

№ п/п	Фактор	Зміст загрози	Можлива реакція компанії
1	Збільшення конкурентів	Впровадження конкурентами аналогічного функціоналу	Основні активності на ринку котрий тримає компанія, збереження існуючих клієнтів.

Продовження таблиці 5.6

№ п/п	Фактор	Зміст загрози	Можлива реакція компанії
2	Складність користування платформою	Втрата попиту через складність системи.	Спрощення системи, нові відділи для конфігурації системи замість замовника.
3	Відсутність попиту	Втрата попиту для нових чи поточних клієнтів.	Реалізація нового функціоналу та покращення роботи старого.

Таблиця 5.7

Фактори можливостей

№ п/п	Фактор	Зміст можливості	Можлива реакція компанії
1	Вхід на ринок обслуговування широкого спектру товарів	Запуск товарів широкого спектру, не тільки транспортних засобів.	Розширення та подальша інтеграція платформи для підтримки нових товарів.
2	Розширення джерел монетизації	Потенційне збільшення шляхів прибутку компанії за допомогою введення нових методів монетизації.	Збільшення кількості реклами, додавання різноманітних платних послуг.

Тепер здійснимо опис загальних рис конкуренції на ринку на основі таблиці 5.8.

Таблиця 5.8

Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)
1. Тип конкуренції - глобальна	Конкуренція компаній які надають платформи для продажу товарів.	Розширення обслуговування клієнтів на міжнародному рівні.
2. За рівнем конкурентної боротьби - міжнародна	Конкуренція компаній охоплює ринки усіх країн світу.	Локалізація продукту для різних країн, що збільшить спектр можливих ринкових опцій.
3. За галузевою ознакою - внутрішньогалузева	Продукти компаній орієнтовані на діяльність інтернет продажів.	Зосередження діяльності на окремих галузях ринку.
4. Конкуренція за видами товарів - послуго-видова	Конкуренція лише за програмні продукти.	Розширення галузей діяльності системи.

На основі аналізу наданої інформації, можемо виділити актуальність реалізації компанії та продукту.

Нижче наведений SWOT аналіз проекту.

Таблиця 5.9

SWOT-аналіз стартап проекту

Сильні сторони: - Динамічна конфігурація платформи в режимі реального часу; - Розробка нового функціоналу специфічного для конкретного клієнтського рішення; - Гнучке розгортання інфраструктури;	Слабкі сторони: - Складність продукту; - Складність локалізації платформи через велику кількість динамічних частин;
Можливості: - Можливість збільшення ринку користувачів для клієнтів, котрі купують послуги компанії; - Збільшення кошт компанії за рахунок продажу додаткових послуг;	Загрози: - Падіння ринку продажу транспортних засобів традиційними способами.

5.4. Розробка ринкової стратегії продукту

Можемо виділити основних користувачів платформою – це малий та середній бізнес, котрий хоче розширити свій функціонал за рахунок інтернет продажів транспортної техніки.

У нижче наведеній таблиці, можемо виділити основні цільові групи понетційних споживачів.

Таблиця 5.10

Вибір цільових груп потенційних споживачів

№ п/п	Опис профілю цільової групи потенційних клієнтів	Готовність споживачі в сприйняти продукт	Орієнтовний попит в межах цільової групи (сегменту)	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
1	Дилери та автомобільні салони.	Середня	Вище середнього	Середня	Середня
2	Малий та середній бізнес, котрий спеціалізуєтсья на продажі товарів.	Висока	Високий	Середня	Середня

5.5. Розробка маркетингової програми стартап проекту

Нижче наведена таблиця описує цінові межі, розраховувані для виставлення ціни на продукт.

Таблиця 5.11

Визначення меж встановлення ціни

№ п/п	Рівень цін на товари замінники	Рівень цін на товари-аналоги	Рівень доходів цільової групи споживачів	Верхня та нижня межі встановлення ціни на товар/послугу
1	-	150-200 дол/міс	Високий	Ціна залежатиме від складності розробки та реалізації функціоналу.

У таблиці 5.12 наведено оптимальну систему збуту.

Таблиця 5.12

Формування системи збуту

№ п/п	Специфіка закупівельної поведінки цільових клієнтів	Функції збуту, які має виконувати постачальник товару	Глибина каналу збуту	Оптимальна система збуту
1	Бажання до продажу транспортних засобів в інтернеті з можливістю врахування специфічних бізнес потреб користувача	Торгівля у вигляді місячної або річної підписки	Однорівневий канал збуту	Система збуту власними ресурсами

Висновок до п'ятого розділу

Резул ьтатом виконання даного розділу був аналіз програмного продукту як стартапу. Основна ідея якого полягає у реалізації динамічної платформи для надання малому та середньому бізнесу можливості продажу транспортної техніки. Рішення може бути специфічним під конкретного клієнта.

Був проведений аналіз основних ринкових можливостей запуску стартап проекту, із зазначенням характеристик потенційного ринку, клієнтів, та різних факторів.

Також була розроблена ринкова стратегія продукту, формалізована за допомогою таблиць із вибірками груп потенційних споживачів та базової стратегії конкурентної поведінки.

В кінці аналізу, було наведено маркетингову програму стартап проекту.

Отже, можна зробити висновок, що даний продукт є цілком актуальним на ринку, має переваги перед схожими аналогами.

ВИСНОВКИ

На основі виконання даної магістерської дисертації, я реалізував B2B рішення системи для трейдингу транспортними засобами, що дозволяє динамічно розгорнути платформу під конкретного клієнта. Клієнт має можливість налаштувати платформу під свої потреби та оновлювати її в режимі роботи.

Додаток був спроектований та реалізований на основі мікросервісної архітектури. Платформа має набір ізольованих мікросервісів, кожен з яких відповідає за свою ділянку роботи, із підтримкою горизонтального масштабування необхідних компонентів.

Кожен з компонентів реалізований мовою програмування Java, із використанням фреймворку Spring, та його рішенням Spring Cloud, для забезпечення мікросервісного підходу.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

[1] Офіційна документація Spring [Електронний ресурс]. – 2012. – Режим доступу до ресурсу: https://spring.io/docs.

[2] Spring Recipes: A Problem-Solution Approach – London: Apress, 2012. – 2312 с.

[3] MySQL Database [Електронний ресурс]. – 2017. – Режим доступу до ресурсу: https://dev.mysql.com/doc/.

[4] Eureka Server [Електронний ресурс]. – 2015. – Режим доступу до ресурсу: https://cloud.spring.io/spring-cloud-netflix/multi/multi_spring-cloud-eureka-server.html.

[5] Офіційна документація Spring Cloud [Електронний ресурс]. – 2018. – Режим доступу до ресурсу: https://spring.io/projects/spring-cloud.

[6] Офіційна документація Apache Kafka [Електронний ресурс]. – 2010. – Режим доступу до ресурсу: https://kafka.apache.org/documentation/.

[7] Open Feign [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://spring.io/projects/spring-cloud-openfeign.

[8] Офіційна документація Docker [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://docs.docker.com/.

[9] Autobahn-tech [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: http://autobahn.tech/.

[10] Wizzle [Електронний ресурс]. – 2017. – Режим доступу до ресурсу: https://www.wizzle.co.uk/.

[11] Carwago [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://carvago.com/.

[12] ADESA [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://www.adesa.eu/.

[13] Zig Wheels [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://www.zigwheels.com/.

[14] SwapMotors [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://www.swapmotors.com/.

[15] Car Wow [Електронний ресурс]. – 2018. – Режим доступу до ресурсу: https://www.carwow.co.uk/.

[16] Car Some [Електронний ресурс]. – 2016. – Режим доступу до ресурсу: https://www.carsome.my/.

[17] Microservice architecture patterns [Електронний ресурс]. – 2015. – Режим доступу до ресурсу: https://microservices.io/patterns/microservices.html/.

[18] Офіційна документація Hibernate [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://hibernate.org/orm/documentation/5.4/.

[19] Amazon Web Services [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://aws.amazon.com/.

[20] Служба хмарних обчислень Azure [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://azure.microsoft.com/.

[21] Офіційна документація EclipseLink [Електронний ресурс]. – 2016. – Режим доступу до ресурсу: https://www.eclipse.org/eclipselink/.

[22] AMPQ Overview [Електронний ресурс]. – 2017. – Режим доступу до ресурсу: https://www.amqp.org/.

[23] Офіційна документація Zuul [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://zuul-ci.org/docs/zuul/.

[24] Zuul API [Електронний ресурс]. – 2018. – Режим доступу до ресурсу: http://netflix.github.io/zuul/javadoc/zuul-core/index.html?help-doc.html /.

[25] Spring Cloud Projects [Електронний ресурс]. – 2016. – Режим доступу до ресурсу: https://projects.spring.io/spring-cloud/spring-cloud.html/.

ДОДАТОК А

Лістинг додатку платформи трейдингу транспортними засобами

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.alokhin.trading-platform</groupId>
    <artifactId>trading-platform</artifactId>
    <version>0.0.1-SNAPSHOT</version>

    <name>trading-platform</name>
    <description>Trading-platform</description>

    <packaging>pom</packaging>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

        <build.manifest.section>Build Details</build.manifest.section>
        <build.unknown>UNKNOWN</build.unknown>
        <build.number>${build.unknown}</build.number>
        <build.revision>${build.unknown}</build.revision>
        <health.endpoint>unknown</health.endpoint>
        <checkstyleConfig>checkstyle.xml</checkstyleConfig>

        <surefire.version>3.0.0-M3</surefire.version>
        <failsafe.version>3.0.0-M3</failsafe.version>
        <release.plugin.version>2.5.3</release.plugin.version>
        <maven-compiler-plugin.version>3.8.0</maven-compiler-plugin.version>
        <maven-checkstyle-plugin.version>3.0.0</maven-checkstyle-plugin.version>
        <maven-jar-plugin.version>2.4</maven-jar-plugin.version>
        <jacoco-maven-plugin.version>0.8.3</jacoco-maven-plugin.version>
        <liquibase-maven-plugin.version>3.6.3</liquibase-maven-plugin.version>
        <jupiter-engine.version>5.6.2</jupiter-engine.version>

        <java.version>14</java.version>
```

```
    <spring-boot.version>2.3.1.RELEASE</spring-boot.version>
    <postgresql.version>42.2.14</postgresql.version>
    <swagger.version>2.9.2</swagger.version>
    <commons-io.version>2.6</commons-io.version>
    <mapstruct.version>1.3.0.Final</mapstruct.version>
    <testcontainers.version>1.11.4</testcontainers.version>
    <apt.plugin.version>1.1.3</apt.plugin.version>
    <awaitility.version>3.1.6</awaitility.version>
</properties>

<modules>
    <module>trading-platform-service</module>
</modules>

<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-dependencies</artifactId>
            <version>${spring-boot.version}</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-configuration-processor</artifactId>
            <version>${spring-boot.version}</version>
            <optional>true</optional>
            <exclusions>
                <exclusion>
                    <groupId>com.vaadin.external.google</groupId>
                    <artifactId>android-json</artifactId>
                </exclusion>
            </exclusions>
        </dependency>
        <dependency>
            <groupId>org.postgresql</groupId>
            <artifactId>postgresql</artifactId>
            <version>${postgresql.version}</version>
        </dependency>
        <dependency>
            <groupId>commons-io</groupId>
            <artifactId>commons-io</artifactId>
            <version>${commons-io.version}</version>
        </dependency>
```

```
<dependency>
    <groupId>org.mapstruct</groupId>
    <artifactId>mapstruct</artifactId>
    <version>${mapstruct.version}</version>
</dependency>
<dependency>
    <groupId>org.mapstruct</groupId>
    <artifactId>mapstruct-processor</artifactId>
    <version>${mapstruct.version}</version>
</dependency>
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-bean-validators</artifactId>
    <version>${swagger.version}</version>
</dependency>
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger2</artifactId>
    <version>${swagger.version}</version>
</dependency>
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger-ui</artifactId>
    <version>${swagger.version}</version>
</dependency>
<dependency>
    <groupId>org.awaitility</groupId>
    <artifactId>awaitility</artifactId>
    <version>${awaitility.version}</version>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter</artifactId>
    <version>${jupiter-engine.version}</version>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.testcontainers</groupId>
    <artifactId>testcontainers</artifactId>
    <version>${testcontainers.version}</version>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.testcontainers</groupId>
```

```
                <artifactId>postgresql</artifactId>
                <version>${testcontainers.version}</version>
                <scope>test</scope>
            </dependency>
            <dependency>
                <groupId>org.testcontainers</groupId>
                <artifactId>junit-jupiter</artifactId>
                <version>${testcontainers.version}</version>
                <scope>test</scope>
            </dependency>
        </dependencies>
    </dependencyManagement>

    <dependencies>
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <scope>provided</scope>
        </dependency>
    </dependencies>

    <build>
        <pluginManagement>
            <plugins>
                <plugin>
                    <groupId>com.mysema.maven</groupId>
                    <artifactId>apt-maven-plugin</artifactId>
                    <version>${apt.plugin.version}</version>
                    <executions>
                        <execution>
                            <goals>
                                <goal>process</goal>
                            </goals>
                            <configuration>
                                <outputDirectory>target/generated-
sources/java</outputDirectory>

<processor>com.querydsl.apt.jpa.JPAAnnotationProcessor</processor>
                            </configuration>
                        </execution>
                    </executions>
                </plugin>
                <plugin>
                    <groupId>org.apache.maven.plugins</groupId>
                    <artifactId>maven-release-plugin</artifactId>
```

```
                    <version>${release.plugin.version}</version>
                    <configuration>
                        <goals>install deploy</goals>
                        <!--                        <tag>${releaseVersion}</tag>-->
                        <arguments>-DskipTests=true -s settings.xml</arguments>
                    </configuration>
                </plugin>
                <plugin>
                    <groupId>org.apache.maven.plugins</groupId>
                    <artifactId>maven-compiler-plugin</artifactId>
                    <version>${maven-compiler-plugin.version}</version>
                    <configuration>
                        <release>${java.version}</release>
                    </configuration>
                </plugin>
                <plugin>
                    <groupId>org.apache.maven.plugins</groupId>
                    <artifactId>maven-checkstyle-plugin</artifactId>
                    <version>${maven-checkstyle-plugin.version}</version>
                    <executions>
                        <execution>
                            <id>checkstyle</id>
                            <phase>validate</phase>
                            <goals>
                                <goal>check</goal>
                            </goals>
                            <configuration>
                                <failOnViolation>true</failOnViolation>
                                <configLocation>checkstyle.xml</configLocation>
                                <sourceDirectories>

<sourceDirectory>${project.build.sourceDirectory}</sourceDirectory>
                                </sourceDirectories>
                            </configuration>
                        </execution>
                    </executions>
                </plugin>
                <plugin>
                    <groupId>org.apache.maven.plugins</groupId>
                    <artifactId>maven-jar-plugin</artifactId>
                    <version>${maven-jar-plugin.version}</version>
                    <configuration>
                        <archive>
                            <manifest>
                                <addClasspath>true</addClasspath>
```

```
                            </manifest>
                            <manifestEntries>
                                <Build-Time>${maven.build.timestamp}</Build-Time>
                            </manifestEntries>
                            <manifestSections>
                                <manifestSection>
                                    <name>${build.manifest.section}</name>
                                    <manifestEntries>
                                        <Implementation-Title>${project.name}</Implementation-
Title>
                                        <Implementation-
Version>${project.version}</Implementation-Version>
                                        <Implementation-Build-
Number>${build.number}</Implementation-Build-Number>
                                    </manifestEntries>
                                </manifestSection>
                            </manifestSections>
                        </archive>
                    </configuration>
                </plugin>
                <plugin>
                    <artifactId>maven-surefire-plugin</artifactId>
                    <version>${surefire.version}</version>
                    <configuration>
                        <useSystemClassLoader>false</useSystemClassLoader>
                        <includes>
                            <include>*.java</include>
                        </includes>
                    </configuration>
                </plugin>
                <plugin>
                    <groupId>org.jacoco</groupId>
                    <artifactId>jacoco-maven-plugin</artifactId>
                    <version>${jacoco-maven-plugin.version}</version>
                    <executions>
                        <execution>
                            <id>default-prepare-agent</id>
                            <goals>
                                <goal>prepare-agent</goal>
                            </goals>
                        </execution>
                        <execution>
                            <id>default-report</id>
                            <phase>prepare-package</phase>
                            <goals>
```

```
                                <goal>report</goal>
                            </goals>
                        </execution>
                    </executions>
                </plugin>
            </plugins>
        </pluginManagement>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-checkstyle-plugin</artifactId>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-jar-plugin</artifactId>
            </plugin>
            <plugin>
                <artifactId>maven-surefire-plugin</artifactId>
            </plugin>
            <plugin>
                <groupId>org.jacoco</groupId>
                <artifactId>jacoco-maven-plugin</artifactId>
            </plugin>
            <plugin>
                <groupId>org.liquibase</groupId>
                <artifactId>liquibase-maven-plugin</artifactId>
                <version>${liquibase-maven-plugin.version}</version>
                <configuration>
                    <propertyFile>src/main/resources/liquibase.properties</propertyFile>
                </configuration>
            </plugin>
        </plugins>
    </build>

</project>

package com.alokhin.trading-platform;

import com.alokhin.trading-platform.property.FileProperties;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
```

```
import org.springframework.boot.context.properties.EnableConfigurationProperties;

@EnableConfigurationProperties(FileProperties.class)
@SpringBootApplication
public class Trading-platformApplication {

    public static void main(String[] args) {
        SpringApplication.run(Trading-platformApplication.class, args);
    }
}

package com.alokhin.trading-platform.facade;

import com.alokhin.trading-platform.entity.File;
import com.alokhin.trading-platform.entity.StorageType;
import com.alokhin.trading-platform.mapper.FileMapper;
import com.alokhin.trading-platform.property.FileProperties;
import com.alokhin.trading-platform.service.FileService;
import com.alokhin.trading-platform.service.StorageService;
import com.alokhin.trading-platform.util.FileUtils;
import com.alokhin.trading-platform.util.FileValidationUtils;
import com.alokhin.trading-platform.web.dto.FileDto;
import com.alokhin.trading-platform.web.dto.FileMetaDataDto;
import lombok.RequiredArgsConstructor;
import org.mapstruct.factory.Mappers;
import org.springframework.stereotype.Component;
import org.springframework.transaction.annotation.Transactional;
import org.springframework.web.multipart.MultipartFile;

import                                  static                                  com.alokhin.trading-
platform.util.FileValidationUtils.validatePropertyNonNullability;
import static org.apache.commons.io.FilenameUtils.getExtension;
import static org.apache.commons.io.FilenameUtils.removeExtension;

@Component
@RequiredArgsConstructor
public class FileFacade {

    private static final FileMapper MAPPER = Mappers.getMapper(FileMapper.class);

    private final StorageService fileSystemStorageService;
    private final FileService fileService;
    private final FileProperties fileProperties;

    @Transactional
```

```
    public FileMetaDataDto saveNew(final MultipartFile multipartFile) {
        final var contentType = multipartFile.getContentType();
        final var fileName = multipartFile.getResource().getFilename();
        final var bucket = fileProperties.getBucket();

        validatePropertyNonNullability(fileName, "File must have a filename");
        validatePropertyNonNullability(contentType, "File must have a content type");

        FileValidationUtils.validateContentType(contentType);

        final var file = fileService.save(
            File.builder()
                .name(removeExtension(fileName))
                .extension(getExtension(fileName))
                .bucket(bucket)
                .contentType(contentType)
                .storageType(StorageType.FILE_SYSTEM)
                .build()
        );
        fileSystemStorageService.store(bucket, multipartFile);

        return MAPPER.toMetaDataDto(file);
    }

    @Transactional(readOnly = true)
    public FileDto findByGuid(final String guid) {
        final var file = fileService.findByGuid(guid);
        final var fileName = FileUtils.fetchFileName(file.getName(), file.getExtension());
        final                       var                       resource                       =
fileSystemStorageService.loadAsResource(fileProperties.getBucket(), fileName);
        return MAPPER.toDto(file, resource);
    }
}

package com.alokhin.trading-platform.facade;

import com.alokhin.trading-platform.web.dto.CreateUserDto;
import com.alokhin.trading-platform.entity.User;
import com.alokhin.trading-platform.service.UserService;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Component;
import org.springframework.transaction.annotation.Transactional;

import java.util.UUID;
```

```
@Component
@RequiredArgsConstructor
public class UserFacade {

    private final UserService userService;

    @Transactional
    public void signUp(final CreateUserDto dto) {
        final var user = User.builder()
            .uuid(UUID.randomUUID().toString())
            .phone(dto.getPhone())
            .details(dto.getDetails())
            .build();
        userService.save(user);
    }
}

package com.alokhin.trading-platform.service;

import com.alokhin.trading-platform.entity.User;
import com.alokhin.trading-platform.repository.UserRepository;
import com.alokhin.trading-platform.repository.filter.QUserFilter;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import javax.persistence.EntityNotFoundException;

import static java.lang.String.format;

@Service
@RequiredArgsConstructor
public class UserService {

    private final UserRepository userRepository;

    @Transactional(readOnly = true)
    public User findUserById(final Long id) {
        return this.findById(id);
    }

    @Transactional(readOnly = true)
    public User findUserByPhone(final String phone) {
        final var p = QUserFilter.builder().phone(phone).toPredicate();
        return userRepository.findOne(p)
```

```
            .orElseThrow(() -> new EntityNotFoundException(format("Couldn't find User by phone
= %s", phone)));
    }

    @Transactional
    public User save(final User user) {
        return userRepository.save(user);
    }

    private User findById(final Long id) {
        final var p = QUserFilter.builder().id(id).toPredicate();
        return userRepository.findOne(p)
            .orElseThrow(() -> new EntityNotFoundException(format("Couldn't find User by id
= %d", id)));
    }
}
```

```
package com.alokhin.trading-platform.service;

import org.springframework.core.io.Resource;
import org.springframework.web.multipart.MultipartFile;

public interface StorageService {

    void store(final String bucket, final MultipartFile file);

    Resource loadAsResource(final String bucket, final String fileName);
}
```

```
package com.alokhin.trading-platform.service.impl;

import com.alokhin.trading-platform.exception.InvalidArgumentException;
import com.alokhin.trading-platform.exception.StorageException;
import com.alokhin.trading-platform.exception.StorageFileNotFoundException;
import com.alokhin.trading-platform.service.StorageService;
import lombok.RequiredArgsConstructor;
import org.apache.commons.io.FileUtils;
import org.springframework.core.io.Resource;
import org.springframework.core.io.UrlResource;
import org.springframework.stereotype.Service;
import org.springframework.util.StringUtils;
import org.springframework.web.multipart.MultipartFile;

import java.io.IOException;
import java.io.InputStream;
```

```
import java.net.MalformedURLException;
import java.nio.file.Paths;

import static java.lang.String.format;

@Service
@RequiredArgsConstructor
public class FileSystemStorageService implements StorageService {

    @Override
    public void store(final String bucket, final MultipartFile file) {
        if (file.isEmpty()) {
            throw new InvalidArgumentException("Failed to store empty file");
        }
        String fileName = StringUtils.cleanPath(file.getOriginalFilename());
        if (StringUtils.isEmpty(fileName)) {
            throw new InvalidArgumentException("File name is empty");
        }
        try {
            if (fileName.contains("..")) {
                // This is a security check
                throw new InvalidArgumentException(format("Cannot store file with relative path
" +
                    "outside current directory = %s", fileName));
            }
            try (final InputStream inputStream = file.getInputStream()) {
                FileUtils.copyInputStreamToFile(inputStream,
                    Paths.get(bucket, fileName).toFile());
            }
        } catch (final IOException e) {
            throw new StorageException(format("Failed to store file = %s", e));
        }
    }

    @Override
    public Resource loadAsResource(final String bucket, final String fileName) {
        try {
            final var file = Paths.get(bucket, fileName);
            Resource resource = new UrlResource(file.toUri());
            if (resource.exists() || resource.isReadable()) {
                return resource;
            } else {
                throw  new  StorageFileNotFoundException(format("Could  not  read  file:  %s",
fileName));
            }
```

```
        } catch (final MalformedURLException e) {
            throw new StorageFileNotFoundException(format("Could not read file: %s", fileName),
e);
        }
    }
}

package com.alokhin.trading-platform.service;

import com.alokhin.trading-platform.entity.Role;
import com.alokhin.trading-platform.repository.RoleRepository;
import com.alokhin.trading-platform.repository.filter.QRoleFilter;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import javax.persistence.EntityNotFoundException;

import static java.lang.String.format;

@Service
@RequiredArgsConstructor
public class RoleService {

    private final RoleRepository roleRepository;

    @Transactional(readOnly = true)
    public Role findRoleById(final Long id) {
        return this.findById(id);
    }

    @Transactional(readOnly = true)
    public Role findRoleByName(final String name) {
        final var p = QRoleFilter.builder().name(name).toPredicate();
        return roleRepository.findOne(p)
            .orElseThrow(() -> new EntityNotFoundException(format("Couldn't find Role by name
= %s", name)));
    }

    @Transactional
    public Role save(final Role role) {
        return roleRepository.save(role);
    }

    private Role findById(final Long id) {
```

```
        final var p = QRoleFilter.builder().id(id).toPredicate();
        return roleRepository.findOne(p)
            .orElseThrow(() -> new EntityNotFoundException(format("Couldn't find Role by id
= %d", id)));
    }
}

package com.alokhin.trading-platform.repository;

import com.alokhin.trading-platform.entity.User;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.querydsl.QuerydslPredicateExecutor;
import org.springframework.stereotype.Repository;

@Repository
public interface UserRepository extends JpaRepository<User, Long>,
QuerydslPredicateExecutor<User> {
}

package com.alokhin.trading-platform.repository;

import com.alokhin.trading-platform.entity.Role;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.querydsl.QuerydslPredicateExecutor;
import org.springframework.stereotype.Repository;

@Repository
public interface RoleRepository extends JpaRepository<Role, Long>,
QuerydslPredicateExecutor<Role> {
}

package com.alokhin.trading-platform.repository.filter;

import com.alokhin.trading-platform.entity.QFile;
import com.querydsl.core.types.Predicate;
import com.querydsl.core.types.dsl.BooleanExpression;

public final class QFileFilter {

    private BooleanExpression pred;

    private QFileFilter() {
        this.pred = file().active.eq(Boolean.TRUE);
    }
```

```
    public static QFileFilter builder() {
        return new QFileFilter();
    }

    public Predicate toPredicate() {
        return this.pred;
    }

    public QFileFilter guid(final String guid) {
        if (guid != null) {
            this.pred = this.pred.and(file().guid.eq(guid));
        }
        return this;
    }

    private static QFile file() {
        return QFile.file;
    }
}

package com.alokhin.trading-platform.repository.filter;

import com.alokhin.trading-platform.entity.QRole;
import com.querydsl.core.types.Predicate;
import com.querydsl.core.types.dsl.BooleanExpression;
import org.apache.commons.lang3.StringUtils;

public final class QRoleFilter {

    private BooleanExpression pred;

    private QRoleFilter() {
        this.pred = role().active.eq(Boolean.TRUE);
    }

    public static QRoleFilter builder() {
        return new QRoleFilter();
    }

    public Predicate toPredicate() {
        return this.pred;
    }

    public QRoleFilter id(final Long id) {
        if (id != null) {
```

```
            this.pred = this.pred.and(role().id.eq(id));
        }
        return this;
    }

    public QRoleFilter name(final String name) {
        if (StringUtils.isNotEmpty(name)) {
            this.pred = this.pred.and(role().name.eq(name));
        }
        return this;
    }

    private static QRole role() {
        return QRole.role;
    }
}

package com.alokhin.trading-platform.repository.filter;

import com.alokhin.trading-platform.entity.QUser;
import com.querydsl.core.types.Predicate;
import com.querydsl.core.types.dsl.BooleanExpression;
import org.apache.commons.lang3.StringUtils;

public final class QUserFilter {

    private BooleanExpression pred;

    private QUserFilter() {
        this.pred = user().active.eq(Boolean.TRUE);
    }

    public static QUserFilter builder() {
        return new QUserFilter();
    }

    public Predicate toPredicate() {
        return this.pred;
    }

    public QUserFilter id(final Long id) {
        if (id != null) {
            this.pred = this.pred.and(user().id.eq(id));
        }
        return this;
```

```
    }

    public QUserFilter phone(final String phone) {
        if (StringUtils.isNotEmpty(phone)) {
            this.pred = this.pred.and(user().phone.eq(phone));
        }
        return this;
    }

    private static QUser user() {
        return QUser.user;
    }
}
```

```
package com.alokhin.trading-platform.web.controller;

import com.alokhin.trading-platform.web.dto.CreateUserDto;
import com.alokhin.trading-platform.facade.UserFacade;
import lombok.RequiredArgsConstructor;
import org.springframework.validation.annotation.Validated;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseStatus;
import org.springframework.web.bind.annotation.RestController;

import javax.validation.Valid;

import static org.springframework.http.HttpStatus.CREATED;

@Validated
@RequiredArgsConstructor
@RestController
@RequestMapping(UserV1Controller.ROOT)
public class UserV1Controller {

    public static final String ROOT = "/v1/users";

    private final UserFacade userFacade;

    @PostMapping
    @ResponseStatus(CREATED)
    public void signUp(@Valid @RequestBody final CreateUserDto dto) {
        userFacade.signUp(dto);
    }
```

```
}

package com.alokhin.trading-platform.web.controller;

import com.alokhin.trading-platform.facade.FileFacade;
import com.alokhin.trading-platform.util.WebResponseUtils;
import com.alokhin.trading-platform.web.dto.FileMetaDataDto;
import lombok.RequiredArgsConstructor;
import org.springframework.core.io.Resource;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.validation.annotation.Validated;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestPart;
import org.springframework.web.bind.annotation.ResponseStatus;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.multipart.MultipartFile;

import static org.springframework.http.HttpStatus.CREATED;

@Validated
@RequiredArgsConstructor
@RestController
@RequestMapping(FileV1Controller.ROOT)
public class FileV1Controller {

    public static final String ROOT = "/v1/files";

    private final FileFacade fileFacade;

    @PostMapping(value = "/upload", consumes = MediaType.MULTIPART_FORM_DATA_VALUE)
    @ResponseStatus(CREATED)
    public FileMetaDataDto upload(@RequestPart("file") MultipartFile file) {
        return fileFacade.saveNew(file);
    }

    @GetMapping("/{guid}")
    public ResponseEntity<Resource> findFile(@PathVariable("guid") final String guid) {
        final var file = fileFacade.findByGuid(guid);
        return WebResponseUtils.resourceResponse(file);
    }
}
```